CAIRO SECURITY CLAN

PARADEX

SECURITY ASSESMENT REPORT

MAY 2025

Prepared for

PARADEX

Contents

1 About Cairo Security Clan **2**

2 Disclaimer **2**

3 Executive Summary **3**

4 Overview **4**
4.1 Assumptions . . . 4

5 Summary of Audit **5**
5.1 Scoped Files . . . 5
5.2 Issues . . . 6

6 Risk Classification **7**

7 Issues by Severity Levels **8**
7.1 Critical . . . 8
7.1.1 Oracle Could Be Re-initialized . . . 8
7.1.2 `account_transfer_partial(...)` Is Missing Access Control And Does Not Check Transfer Restrictions . . . 8
7.2 High . . . 9
7.2.1 `erc20._transfer(...)` Logic Should Be Changed . . . 9
7.2.2 `fee_commission` Not Deducted From The `fee` . . . 10
7.2.3 Socialized Loss Not Accounted For Except During Withdrawal . . . 11
7.2.4 Liquidate Partial Transfer Incorrect Amount Of Collateral To Liquidator And Insurance Account . . . 12
7.3 Medium . . . 13
7.3.1 Owner Can Withdraw Nearly The Entire Initial Deposit . . . 13
7.3.2 Owner's Share Calculation Inaccurate Due To Use Of Current Share Supply Instead Of Post-Redeem Share Supply . . . 13
7.3.3 Centralized Risks . . . 14
7.3.4 Allowing Negative Fee Rates Could Lead To Losses For Fee Account . . . 15
7.3.5 Vault Is Never A Caller In Registry Contract . . . 16
7.3.6 Function `_transfer_positions_internal()` Fails To Create Asset Balance For Receiver If Asset Was Not Previously Used . . . 16
7.3.7 Function `is_risky()` Does Not Account For Trading Fee When Checking If The Trade Is Risky Or Not . . . 18
7.3.8 Closed Vault Cannot Transfer Assets to the Auxiliary Account . . . 18
7.4 Low . . . 19
7.4.1 Profit Shares Should Round Up . . . 19
7.4.2 `on_receive(...)` Does Not Create an Account . . . 19
7.4.3 Users Could Add Duplicate Account Addresses to the Protocol . . . 20
7.4.4 `taker_fee` or `maker_fee` Can Potentially Underflow the Balance . . . 21
7.4.5 Missing Validations in Function `settleTrade()` . . . 22
7.5 Informational . . . 23
7.5.1 The Function Name Does Not Reflect The Returned Data . . . 23
7.5.2 Sub-operator Cannot Be Registered . . . 23
7.5.3 Disallow Withdrawals Which Are Below Scaling Factor . . . 24
7.5.4 `liquidate` Should Calculate Penalty Settlement Value Too . . . 24
7.5.5 `settle_market(...)` Can Potentially Lead To Account Bankruptcy . . . 25
7.5.6 `liquidate_partial(...)` Could Check If The Partial Liquidation Made An Account Healthy . . . 25
7.5.7 Incorrect Value Passed To `is_liquidation` When Calling `transfer_internal()` In The `account_transfer_partial()` Function . . . 26
7.5.8 SRC5 External Function Not Exposed . . . 26
7.5.9 Unused Code . . . 27
7.6 Best Practices . . . 28
7.6.1 Percentages Should Have Upper Bound . . . 28
7.6.2 Unnecessary Multiplication with WAD in Both Numerator and Denominator . . . 28

8 Post Audit Recommendations **29**

9 Test Evaluation **30**
9.1 Compilation Output . . . 30
9.2 Tests Output . . . 30

1 About Cairo Security Clan

Cairo Security Clan is a leading force in the realm of blockchain security, dedicated to fortifying the foundations of the digital age. As pioneers in the field, we specialize in conducting meticulous smart contract security audits, ensuring the integrity and reliability of decentralized applications built on blockchain technology.

At Cairo Security Clan, we boast a multidisciplinary team of seasoned professionals proficient in blockchain security, cryptography, and software engineering. With a firm commitment to excellence, our experts delve into every aspect of the Web3 ecosystem, from foundational layer protocols to application-layer development. Our comprehensive suite of services encompasses smart contract audits, formal verification, and real-time monitoring, offering unparalleled protection against potential vulnerabilities.

Our team comprises industry veterans and scholars with extensive academic backgrounds and practical experience. Armed with advanced methodologies and cutting-edge tools, we scrutinize and analyze complex smart contracts with precision and rigor. Our track record speaks volumes, with a plethora of published research papers and citations, demonstrating our unwavering dedication to advancing the field of blockchain security.

At Cairo Security Clan, we prioritize collaboration and transparency, fostering meaningful partnerships with our clients. We believe in a customer-oriented approach, engaging stakeholders at every stage of the auditing process. By maintaining open lines of communication and soliciting client feedback, we ensure that our solutions are tailored to meet the unique needs and objectives of each project.

Beyond our core services, Cairo Security Clan is committed to driving innovation and shaping the future of blockchain technology. As active contributors to the ecosystem, we participate in the development of emerging technologies such as Starknet, leveraging our expertise to build robust infrastructure and tools. Through strategic guidance and support, we empower our partners to navigate the complexities of the blockchain landscape with confidence and clarity.

In summary, Cairo Security Clan stands at the forefront of blockchain security, blending technical prowess with a client-centric ethos to deliver unparalleled protection and peace of mind in an ever-evolving digital landscape. Join us in safeguarding the future of decentralized finance and digital assets with confidence and conviction.

2 Disclaimer

Disclaimer Limitations of this Audit:

This report is based solely on the materials and documentation provided by you to Cairo Security Clan for the specific purpose of conducting the security review outlined in the Summary of Audit and Scoped Files. The findings presented here may not be exhaustive and may not identify all potential vulnerabilities. Cairo Security Clan provides this review and report on an "as-is" and "as-available" basis. You acknowledge that your use of this report, including any associated services, products, protocols, platforms, content, and materials, occurs entirely at your own risk.

Inherent Risks of Blockchain Technology:

Blockchain technology remains in its developmental stage and is inherently susceptible to unknown risks and vulnerabilities. This review is specifically focused on the smart contract code and does not extend to the compiler layer, programming language elements beyond the reviewed code, or other potential security risks outside the code itself.

Report Purpose and Reliance:

This report should not be construed as an endorsement of any specific project or team, nor does it guarantee the absolute security of the audited smart contracts. No third party should rely on this report for any purpose, including making investment or purchasing decisions.

Liability Disclaimer:

To the fullest extent permitted by law, Cairo Security Clan disclaims all liability associated with this report, its contents, and any related services and products arising from your use. This includes, but is not limited to, implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Third-Party Products and Services:

Cairo Security Clan does not warrant, endorse, guarantee, or assume responsibility for any products or services advertised by third parties within this report, nor for any open-source or third-party software, code, libraries, materials, or information linked to, referenced by, or accessible through this report, its content, and related services and products. This includes any hyperlinked websites, websites or applications appearing on advertisements, and Cairo Security Clan will not be responsible for monitoring any transactions between you and third-party providers. It is recommended that you exercise due diligence and caution when considering any third-party products or services, just as you would with any purchase or service through any medium.

Disclaimer of Advice:

FOR THE AVOIDANCE OF DOUBT, THIS REPORT, ITS CONTENT, ACCESS, AND/OR USE, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHOULD NOT BE CONSIDERED OR RELIED UPON AS FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER PROFESSIONAL ADVICE.

3 Executive Summary

This document presents the security review performed by Cairo Security Clan on the Paradex.

Paradex is a decentralized exchange for perpetual contracts, built on a Layer 2 network. It enables users to trade in both the perpetual and options markets. The platform operates by combining both on-chain and off-chain components. The off-chain component handles tasks such as managing the order book, executing the matching engine, and fetching market data from oracles. Meanwhile, the on-chain component is primarily responsible for settling trades, moving users' funds, and recording position data on the blockchain. Learn more from docs.

The audit was performed using

- manual analysis of the codebase,
- automated analysis tools,
- simulation of the smart contract,
- analysis of edge test cases

30 points of attention, where 2 are classified as Critical, 4 are classified as High, 8 are classified as Medium,5 are classified as Low,9 are classified as Informational and 2 are classified as Best Practices. The issues are summarized in Fig. 1.

This document is organized as follows. Section 1 About Cairo Security Clan. Section 2 Disclaimer. Section 3 Executive Summary. Section 4 Summary of Audit. Section 5 Risk Classification. Section 6 Issues by Severity Levels. Section 7 Test Evaluation.

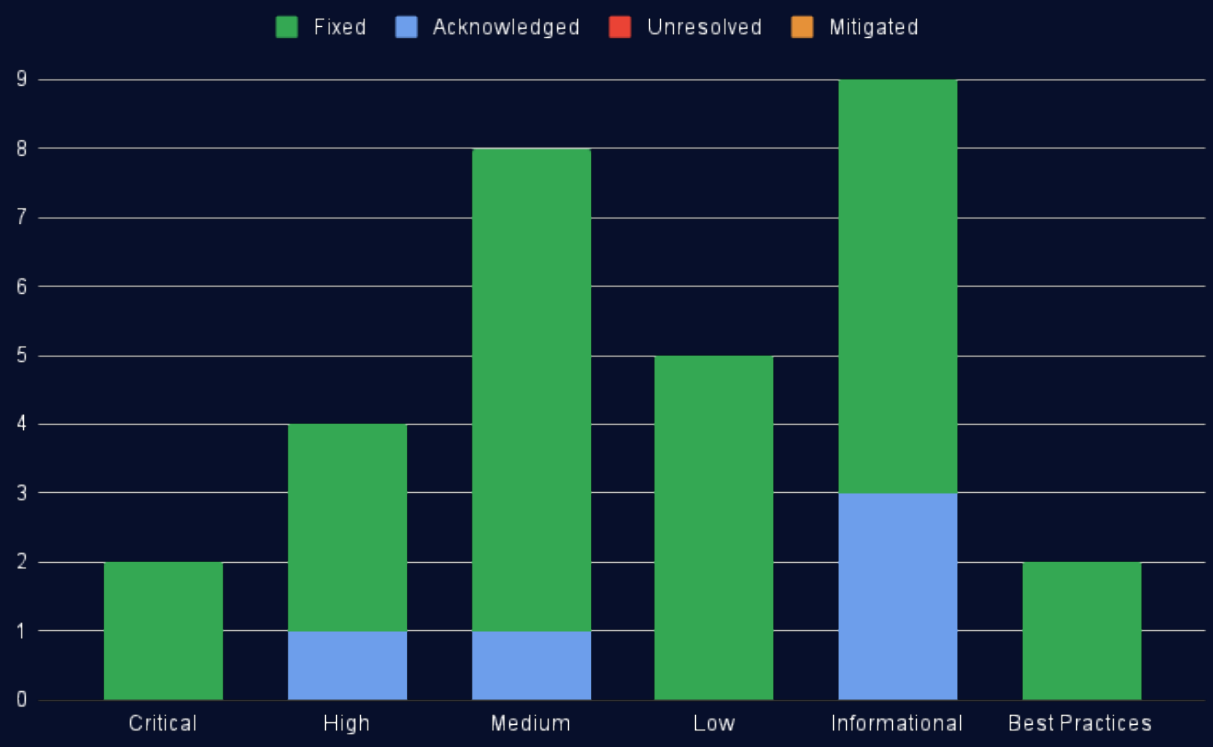

Fig 1: Distribution of issues: Critical (2), **High** (4), **Medium** (8), **Low** (5), **Informational** (9), **Best Practices** (2). **Distribution of status: Fixed** (25), **Acknowledged** (5), **Mitigated** (0), **Unresolved** (0).

4 Overview

Paradex is a decentralized exchange for perpetual contracts, built on a Layer 2 network. It enables users to trade in both the perpetual and options markets. The platform operates by combining both on-chain and off-chain components. The off-chain component handles tasks such as managing the order book, executing the matching engine, and fetching market data from oracles. Meanwhile, the on-chain component is primarily responsible for settling trades, moving users' funds, and recording position data on the blockchain.

Our review primarily focused on the security aspects of the on-chain component, which consists of 5 key smart contracts: Paraclear, Vault, Factory, Registry and Oracle.

Our System Overview

- **Paraclear:** The Paraclear contract serves as the primary perpetual exchange contract, responsible for managing all positions and balance accounting. It enables the executor (a trusted role) to settle trades and positions upon off-chain requests.
- **Vault:** The Vault contract is used to pool users' assets. The vault operator trades these assets on the Paraclear contract. Users receive shares in the vault, which are valued based on the operator's account value within the system.
- **Factory:** The Factory contract facilitates the deployment of new vaults and configures their initial parameters.
- **Registry:** The Registry contract maintains a record of roles within the ecosystem, such as vaults, operators, sub-operators, and auxiliary accounts. It enforces role-specific restrictions on transfers to, from, and within the Paraclear contract.
- **Oracle:** Managed by Paradex, the Oracle contract provides up-to-date pricing for all markets and assets within the Paraclear contract.

4.1 Assumptions

The on-chain component works with several assumptions

- The protocol relies on several privileged roles that introduce significant centralization risks. These roles include:
  - **Executor:** The Executor role is responsible for settling trades, managing positions, and executing liquidations. All related functions can only be called by this role, with no alternative mechanism available for opening, closing, or liquidating positions.
  - **Configurator:** The Configurator role has the authority to modify various parameters of the Paraclear contract. This includes configuring fees, managing account referrals, and setting discounts.
  - **Admin/Owner:** The Admin (or Owner) role has comprehensive control over the protocol. This role can grant or revoke permissions, upgrade contracts, and pause withdrawals, effectively centralizing significant aspects of protocol governance.
- Only works with USDC collateral
- The centralised oracle must aggregate price data from diverse sources to reduce the risk of manipulation. Since all system pricing relies on this oracle, its security and reliability are very important.

5 Summary of Audit

Audit Type	Security Review
Cairo Version	2.8.2
Final Report	12/05/2025
Repository	tradeparadex/contracts
Initial Commit Hash	7a344a865d0d36d0347b88cff799e7ae3cbb9a0e
Final Commit Hash	0eb81b26a67666c399b4e16b39a96c19848ab7fd
Documentation	Website documentation
Test Suite Assessment	High

5.1 Scoped Files

	Contracts
1	oracle/src/interface.cairo
2	oracle/src/lib.cairo
3	oracle/src/oracle.cairo
4	paraclear/src/account/account.cairo
5	paraclear/src/account/interface.cairo
6	paraclear/src/paraclear/interface.cairo
7	paraclear/src/paraclear/math.cairo
8	paraclear/src/paraclear/paraclear.cairo
9	paraclear/src/paraclear/roles.cairo
10	paraclear/src/perpetual/future.cairo
11	paraclear/src/perpetual/interface.cairo
12	paraclear/src/perpetual/option.cairo
13	paraclear/src/perpetual/perpetual_asset.cairo
14	paraclear/src/token/interface.cairo
15	paraclear/src/token/token.cairo
16	paraclear/src/account.cairo
17	paraclear/src/lib.cairo
18	paraclear/src/paraclear.cairo
19	paraclear/src/perpetual.cairo
20	paraclear/src/token.cairo
21	src/lib.cairo
22	vaults/src/factory/factory.cairo
23	vaults/src/factory/interface.cairo
24	vaults/src/mocks/mock_paraclear.cairo
25	vaults/src/mocks/mock_random.cairo
26	vaults/src/mocks/mock_token.cairo
27	vaults/src/paraclear/interface.cairo
28	vaults/src/paraclear/paraclear.cairo
29	vaults/src/registry/interface.cairo
30	vaults/src/registry/registry.cairo
31	vaults/src/vault/vault.cairo
32	vaults/src/vault/interface.cairo
33	vaults/src/factory.cairo
34	vaults/src/lib.cairo
35	vaults/src/mocks.cairo
36	vaults/src/paraclear.cairo
37	vaults/src/registry.cairo
38	vaults/src/utils.cairo
39	vaults/src/vault.cairo

5.2 Issues

	Findings	Severity	Update
1	Oracle could be re-initialized	Critical	Fixed
2	`account_transfer_partial(...)` is missing access control and does not check transfer restrictions	Critical	Fixed
3	`erc20._transfer(...)` logic should be changed	High	Fixed
4	`fee_commission` not deducted from the fee	High	Fixed
5	Socialized loss not accounted for except during withdrawal	High	Acknowledged
6	Liquidate partial transfer incorrect amount of collateral to liquidator and insurance account	High	Fixed
7	Owner can withdraw nearly the entire initial deposit	Medium	Fixed
8	Owner's share calculation inaccurate due to use of current share supply instead of post-redeem share supply	Medium	Fixed
9	Centralized risks	Medium	Acknowledged
10	Allowing negative fee rates could lead to losses for fee account	Medium	Fixed
11	Vault is never a caller in registry contract	Medium	Fixed
12	Function `_transfer_positions_internal()` fails to create asset balance for receiver if asset was not previously used	Medium	Fixed
13	Function `is_risky()` does not account for trading fee when checking if the trade is risky or not	Medium	Fixed
14	Closed vault cannot transfer assets to the auxilary account	Medium	Fixed
15	Profit shares should round up	Low	Fixed
16	`on_receive(...)` does not create an account	Low	Fixed
17	Users could add duplicate account addresses to the protocol	Low	Fixed
18	`taker_fee` or `maker_fee` can potentially underflow the balance	Low	Fixed
19	Missing validations in function `settleTrade()`	Low	Fixed
20	The function name does not reflect the returned data	Informational	Fixed
21	Sub-operator cannot be registered	Informational	Fixed
22	Disallow withdrawals which are below scaling factor	Informational	Fixed
23	`liquidate` should calculate penalty settlement value too	Informational	Fixed
24	`settle_market(...)` can potentially lead to account bankruptcy	Informational	Acknowledged
25	`liquidate_partial(...)` could check if the partial liquidation made an account healthy	Informational	Acknowledged
26	Incorrect value passed to `is_liquidation` when calling `transfer_internal()` in the `account_transfer_partial()` function	Informational	Acknowledged
27	SRC5 external function not exposed	Informational	Fixed
28	Unused Code	Informational	Fixed
29	Percentages should have upper bound	Best Practices	Fixed
30	Unnecessary multiplication with WAD in both enumerator and denominator	Best Practices	Fixed

6 Risk Classification

The risk rating methodology used by Cairo Security Clan follows the principles established by the CVSS risk rating methodology. The severity of each finding is determined by two factors: **Likelihood** and **Impact**.

Likelihood measures how likely an attacker will uncover and exploit the finding. This factor will be one of the following values:

a) **High**: The issue is trivial to exploit and has no specific conditions that need to be met;

b) **Medium**: The issue is moderately complex and may have some conditions that need to be met;

c) **Low**: The issue is very complex and requires very specific conditions to be met.

When defining the likelihood of a finding, other factors are also considered. These can include but are not limited to Motive, opportunity, exploit accessibility, ease of discovery, and ease of exploit.

Impact is a measure of the damage that may be caused if an attacker exploits the finding. This factor will be one of the following values:

a) **High**: The issue can cause significant damage such as loss of funds or the protocol entering an unrecoverable state;

b) **Medium**: The issue can cause moderate damage such as impacts that only affect a small group of users or only a particular part of the protocol;

c) **Low**: The issue can cause little to no damage such as bugs that are easily recoverable or cause unexpected interactions that cause minor inconveniences.

When defining the impact of a finding other factors are also considered. These can include but are not limited to Data/state integrity, loss of availability, financial loss, and reputation damage. After defining the likelihood and impact of an issue, the severity can be determined according to the table below.

		Likelihood		
		High	**Medium**	**Low**
Impact	**High**	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Info/Best Practices

To address issues that do not fit a High/Medium/Low severity, Cairo Security Clan also uses three more finding severities: **Informational**, **Best Practices** and **Gas**

a) **Informational** findings do not pose any risk to the application, but they carry some information that the audit team intends to formally pass to the client;

b) **Best Practice** findings are used when some piece of code does not conform with smart contract development best practices;

c) **Gas** findings are used when some piece of code uses more gas than it should be or have some functions that can be removed to save gas.

7 Issues by Severity Levels

7.1 Critical

7.1.1 Oracle Could Be Re-initialized

File(s): oracle/src/oracle.cairo

Description: The `ParaclearOracle` contract contains an `initializer(...)` function responsible for initializing the `AccessControlComponent` and granting the `0` role to the user specified in the `default_admin` parameter. This function is invoked during the contract's creation within the `constructor`. However, it is part of the `ParaclearOracleImpl` implementation which exposes all of its functions to the public.

```
fn initializer(ref self: ContractState, default_admin: ContractAddress) {
    self.accesscontrol.initializer();
    self.accesscontrol._grant_role(0, default_admin);
}
```

Since the `initializer(...)` function is publicly available and no access control or safeguards (such as a variable to prevent reinitialization) have been implemented, the entire oracle contract is vulnerable to being reinitialized. This could allow an arbitrary user to gain admin privileges and modify the price to any arbitrary value.

Recommendation(s): Restrict the `initializer(...)` function to internal use or ensure it can only be called once.

Status: Fixed

Update from the client: Fixed in this PR #4. Removed external initializer function.

7.1.2 `account_transfer_partial(...)` Is Missing Access Control And Does Not Check Transfer Restrictions

File(s): paraclear/src/paraclear/paraclear.cairo

Description: The `account_transfer_partial(...)` function allows users to transfer a portion of their perpetual positions and assets to any account. This function is intended to be used by a `Vault` contract, enabling it to transfer a part of the operator's or vault's assets to an auxiliary account.

```
fn account_transfer_partial(
    ref self: ContractState,
    account: ContractAddress,
    receiver: ContractAddress,
    account_share: felt252
) -> felt252 {
   // ...
}
```

However, there are two significant implementation issues with this function:

- **Arbitrary Input for `account`:** The function allows arbitrary values for the `account` parameter, which is critical as it permits unauthorized users to transfer assets and positions to any arbitrary receiver, potentially leading to asset theft.
- **Missing Transfer Restrictions:** The `_detect_transfer_restriction(...)` function, responsible for controlling where assets can be transferred, is not invoked. This omission allows the operator of the vault to bypass restrictions and steal all assets from the vault.

Recommendation(s): Consider implementing the access control check to allow only the `Vault` contract to use such functionality to avoid any thefts of assets.

Status: Fixed

Update from the client: Fixed in this PR #7. Added restrictions on the registry.

7.2 High

7.2.1 `erc20._transfer(...)` Logic Should Be Changed

File(s): vaults/src/vault/vault.cairo

Description: The `Vault` contract manages the deposits and withdrawals of assets for the Paraclear account, functioning similarly to an ERC4626 vault. Upon every deposit, it mints shares to the user, and when the user requests a withdrawal, the corresponding shares are burned. However, the contract contains several issues related to the functionality of transferring shares, resulting in broken features:

1. **Bypassing the lockup period:**
   - The contract implements a lockup period for withdrawals, where a user's average deposit time is updated upon deposit to determine when they can withdraw their assets.
   - However, this mechanism is flawed, as users can transfer their shares to another account, allowing the recipient to bypass the lockup period and withdraw immediately.
2. **Violation of `_ensure_min_owner_share(...)` invariant:**
   - The `_ensure_min_owner_share(...)` function enforces a rule requiring the vault owner to retain a minimum percentage of total shares even after a withdrawal.
   - This restriction is ineffective, as the vault owner can transfer shares to another account, circumventing the limitation entirely.
3. **Incorrect `asset_balances` updates:**
   - When shares are transferred or burned (through public `burn(...)`), the `asset_balances(...)` function is not updated accordingly.
   - This oversight can lead to inaccuracies when calculating profit-sharing fees.

Recommendation(s): Update the internal `_transfer(...)` function to address the issues outlined above:

- Prevent share transfers that bypass the lockup period.
- Ensure the `_ensure_min_owner_share(...)` rule cannot be circumvented via share transfers.
- Accurately update `asset_balances(...)` during share transfers and burns to maintain consistent state and fee calculations.

Status: Fixed

Update from the client: Fixed in this PR #38 and PR #54. Added erc20 hook to validate lockup period and min owner share before token transfer.

7.2.2 `fee_commission` Not Deducted From The `fee`

File(s): paraclear/src/paraclear/paraclear.cairo

Description: Fees are charged for each trade or market settlement using the `get_trade_fee(...)` function. This function computes the fee via `calculate_fee(...)` and applies a discount or commission based on the referral type.

However, the current implementation of the `get_trade_fee(...)` function introduces a significant accounting issue when the `fee_commission` is set. The commission is calculated in the following block:

```
if self.referral.fee_commission != @0 {
    let fee_commission = mul_128(
        fee, (*self.referral.fee_commission).try_into().unwrap()
    );
    return (fee.into(), *self.referral.referrer, fee_commission.try_into().unwrap());
}
```

In this implementation:

- The `fee_commission` is derived from the `fee`, but the `fee` itself is not updated before being returned.
- This results in an accounting discrepancy in settlement functions such as `_settlement_fee_payments(...)`:

```
let (fee, referrer, fee_commission) = account_state
    .get_trade_fee(trade_size, trade_price, false, asset);
let balance_after_fee = pending_token_balance - fee;
self.token.write_asset_balance(account, balance_after_fee, initial_token_balance);
let settlement_token_address = self.getSettlementTokenAsset();

self.token.upsert_asset_balance(referrer, settlement_token_address, fee_commission);
self.token.upsert_asset_balance(fee_account, settlement_token_address, fee);
```

The issue occurs because:

1. The `referrer` receives their commission (`fee_commission`).
2. The `fee_account` receives the full `fee`.
3. Only the `fee` is deducted from `pending_token_balance`.

This results in an inflation of balances, as the commission is effectively being double-counted.

Recommendation(s): Update the `get_trade_fee(...)` function to deduct the `fee_commission` from the `fee` before returning it.

Status: Fixed

Update from the client: Fixed in this PR #9. Fixed total fee by deducting fee commission.

7.2.3 Socialized Loss Not Accounted For Except During Withdrawal

File(s): paraclear/src/paraclear/paraclear.cairo

Description: Currently, socialized losses are only applied when users withdraw funds from the protocol. If the protocol is in a state of insolvency, users' withdrawal amounts are reduced to cover these losses. However, this socialized loss should be accounted for in other key areas:

- `_load_account()` function: This function does not factor in socialized losses when loading token balances. As a result, functions relying on `_load_account()`, such as `getAccountValue()`, may return higher values than the actual amount users can withdraw. This discrepancy affects the Vault's total asset calculation, potentially inflating values and leading users to pay profit shares based on overestimated balances. In reality, after accounting for socialized losses, there may be no profit at all.
- New users depositing during insolvency: New users who deposit funds during a period of insolvency are unfairly burdened with pre-existing losses. For example, if a new user deposits and withdraws immediately, they are still responsible for covering losses incurred prior to their deposit. This creates an unfair situation, which may discourage new deposits and hinder the protocol's recovery.

Recommendation(s):

- Accounting adjustment: Implement a mechanism to ensure socialized losses are only applied to existing funds in the system, excluding new deposits.
- Restrict deposits: During insolvency, consider restricting new deposits to prevent new users from inheriting these losses.
- Update `_load_account()`: Modify the `_load_account()` function to factor in socialized losses when loading token balances, ensuring accurate calculations for all related functions.

Status: Acknowledged

Update from the client: Socialized losses ensures that the exchange is solvent in the extreme case where all users withdraw their funds. If the insurance fund has a shortfall (negative account value), then that loss would be socialized across all users (if they withdraw). This means that users who don't withdraw are not affected. The socialized loss factor is dynamic and can get back to 0% if the insurance fund recovers. In such scenarios, users who didn't withdraw during the problematic period are not affected. This is why token balances are not affected by socialized loss.

7.2.4 Liquidate Partial Transfer Incorrect Amount Of Collateral To Liquidator And Insurance Account

File(s): paraclear/src/paraclear/paraclear.cairo

Description: In the Paraclear protocol, there are two liquidation flows implemented by the functions `liquidate()` and `liquidate_partial()`:

1. `liquidate()` **function:** This function handles the full liquidation of an account by transferring all its assets to the liquidator. During this process, a liquidation penalty is applied, which is transferred to the insurance account. The remaining balance after the penalty is transferred to the liquidator.

```
let liquidator_change_amount = account_balance.amount - liq_penalty_d;
let insurance_fund_change_amount = liq_penalty_d;
```

2. `liquidate_partial()` **function:** As the name suggests, this function is intended to partially liquidate an account, with the degree of liquidation determined by the input parameter `liquidation_share`. The liquidation process should transfer a proportional share of the token balance to both the liquidator and the insurance account, maintaining the same ratio as in the full liquidation. Specifically, the amounts should be scaled by `liquidation_share`.

   However, the current implementation of `liquidate_partial()` transfers only the liquidation penalty to the liquidator instead of a proportionate share of the account's token balance.

```
let liq_penalty_full_value = div_128(
    liq_penalty_full, settlement_token_price.try_into().unwrap());
let liq_penalty = mul_128(
    liq_penalty_full_value, liquidation_share.try_into().unwrap());
self.token.transfer_internal(account, liquidator, self.getSettlementTokenAsset(), liq_penalty.into(), 1 );
```

Recommendation(s): Consider modifying the `liquidate_partial()` function to correctly handle partial liquidations by:

1. Transferring (`account_balance.amount` − `liq_penalty_full_value`) × `liquidation_share` amount of settlement token to the liquidator.
2. Transferring `liq_penalty_full_value` × `liquidation_share` to the insurance account.

This adjustment will ensure that the liquidation flow for partial liquidations is consistent with the logic used in the full liquidation (`liquidate()`) function.

Status: Fixed

Update from the client: Fixed in this PR #39. Removed legacy full liquidation logic, removed distinction between insurance fund and liquidator (they always match)

7.3 Medium

7.3.1 Owner Can Withdraw Nearly The Entire Initial Deposit

File(s): vaults/src/vault/vault.cairo

Description: In Paradex, when a new Vault is created, the owner is required to deposit at least a `min_initial_deposit` amount. Other users are not allowed to deposit until the owner has made their deposit. This ensures that the owner holds at least some share of the Vault before any external users contribute.

```
if !is_owner {
    assert(owner_shares > 0, Errors::NO_OWNER_SHARES);
} else if owner_shares == 0 {
    let factory_address = self.vault_factory.read();
    let factory_dispatcher = IFactoryDispatcher { contract_address: factory_address };
    let multiplier = MATH::pow(10, asset_decimals.into());
    assert(
        assets >= multiplier * factory_dispatcher.min_initial_deposit().into(),
        Errors::BELOW_MIN_INITIAL_DEPOSIT
    );}
```

However, there is currently no mechanism in the `request_withdrawal()` function to ensure that the owner still maintains a sufficient share after the initial deposit. As a result, the owner can withdraw nearly all of their initial deposit immediately after it's made, circumventing the minimum deposit check. This could undermine the purpose of ensuring that the owner retains a minimum share in the Vault.

Recommendation(s): Consider adding a check in the `request_withdrawal()` function to ensure that the owner retains at least the minimum initial deposit amount.

Status: Fixed

Update from the client: Fixed in this PR #40 and this PR #54.

7.3.2 Owner's Share Calculation Inaccurate Due To Use Of Current Share Supply Instead Of Post-Redeem Share Supply

File(s): vaults/src/vault/vault.cairo

Description: In each Vault, the owner is required to hold a specific percentage of the total shares to ensure their incentives align with those of other depositors. When the admin requests a withdrawal or burns their share token, the `_ensure_min_owner_share()` function is invoked to check if the owner's share post-transaction will still meet the minimum required threshold.

```
fn _ensure_min_owner_share(self: @ContractState, caller: ContractAddress, shares: u256) {
    let is_owner = self.owner() == caller;
    if (!is_owner) {
        return;
    }
    let status = self.status.read();
    if (status == VaultStatus::Closed) {
        return;
    }
    let current_shares = self.erc20.balance_of(caller);
    let min_owner_share: u256 = self.min_owner_share_percentage().into()
        * (CONSTANTS::WAD / 100);
    let owner_share_after = (current_shares - shares)
        * CONSTANTS::WAD
        / self._total_supply();
    assert(owner_share_after >= min_owner_share, Errors::MIN_OWNER_SHARE);
}
```

In the current implementation, `owner_share_after` is calculated as the owner's remaining shares (`current_shares - shares`) divided by the current total supply. This calculation does not reflect the fact that the owner's share will be a larger percentage of the vault's total supply once the shares are redeemed or burned.

Recommendation(s): Instead of using the current total supply in the denominator, use the post-redemption total supply. Specifically, after deducting the shares to be redeemed or burned, the new total supply should be used for the calculation of the owner's remaining share percentage.

Status: Fixed

Update from the client: Fixed in this PR #40.

7.3.3 Centralized Risks

File(s): paraclear/src/paraclear/paraclear.cairo

Description: The Paradex protocol consists of two components: the offchain orderbook and the onchain smart contract. The offchain orderbook is responsible for receiving signed orders from traders and matching them. These matched orders are then sent to the onchain smart contract by executors, who actually execute and record the trade. However, there are a few key concerns in the current design that introduce centralized risks:

- **Signature validation offchain:** The signatures of traders are validated in the offchain component, which defeats the purpose of decentralization. In a decentralized protocol, the security of traders should not rely on offchain actors. If the executors behave maliciously or are compromised, they could create and submit arbitrary orders on behalf of any trader, potentially causing financial losses. The signature validation should be done on the onchain smart contract itself to ensure the integrity and authenticity of orders.
- **Centralized oracle:** The onchain smart contract relies on a price oracle to execute trades. However, in the current design, all price data is submitted by a single executor, which introduces a significant risk of centralization. If the executor providing the price data is compromised, manipulated, or behaves maliciously, it could lead to incorrect price feeds and adversely affect the trade execution process, potentially resulting in market manipulation or unfair trades.

Recommendation(s): Consider implementing the following suggestions:

- Move the validation of trader signatures from the offchain orderbook to the onchain smart contract.
- Implement a decentralized price oracle solution to mitigate the risk of relying on a single executor for price data. Alternatively, consider using a multi-sig or committee-based approach for submitting price data, where multiple independent parties are required to sign off on the price before it is accepted by the smart contract.

Status: Acknowledged

Update from the client: We acknowledge the centralization risk on the oracle. For what concern the signature verification we started the process to enforce it but it will require to migrate from EIP-712 to SNIP-12 signatures. This is a big breaking change that will require several changes on all sides, including all clients and SDKs. Initial support is implemented in PR #50, but the finalization of this process will require a long migration plan.

7.3.4 Allowing Negative Fee Rates Could Lead To Losses For Fee Account

File(s): paraclear/src/account/account.cairo

Description: In the Paraclear system, makers and takers can have different fee rates. Although one of them may have a negative fee rate, the combined fee rate for the maker and taker must always be non-negative. This condition ensures that, in the worst case, either the taker pays all the fees or even compensates the maker for part of the fee during a trade, and vice versa. This constraint is enforced in the `setAccountFeeRate()` and `setGlobalFeeRate()` functions, where the `net_fee` (the sum of the maker and taker fees) is checked to ensure it is always non-negative (`net_fee >= 0`).

```
fn set_account_fee_rate(
    ref self: ComponentState<TContractState>,
    account: ContractAddress,
    maker_fee: felt252,
    taker_fee: felt252
) {
    self.assert_only_role(roles::CONFIGURATOR_ROLE);
    let net_fee = maker_fee + taker_fee;
    assert!(
        net_fee.try_into().unwrap() >= 0_i128, "Negative total fee rate is not allowed"
    );
    self
        .Paraclear_account_fee_rate
        .write(account, FeeRate { exists: 1, maker: maker_fee, taker: taker_fee })
}
```

However, there are some scenarios where the fee account could still incur a loss after a trade because of the negative fee rates:

- **Capping trading fees for options:** While the positive fee rate is capped, the negative fee is not, meaning the trader's fee could be capped at a certain value, but the fee they may receive could be unlimited. This imbalance creates a risk where the fee account could end up with a loss if the taker receives more than the capped positive fee.

```
fn calculate_fee(
    self: @PerpetualOptionAsset,
    account_state: @AccountState,
    trade_size: i128,
    trade_price: i128,
    fee_rate: i128
) -> i128 {
    let spot_price = _get_spot_price(account_state, (*self).base_asset);
    let base_fee = mul_128(spot_price, fee_rate);
    let fee_cap = mul_128(trade_price, *account_state.perpetual_options_fee_cap);
    let min_value = min_128(base_fee, fee_cap);
    mul_128(trade_size, min_value)
}
```

- **Traders with custom account fee rates:** Even though the `net_fee` is enforced when the fee rates are set, this does not guarantee that the sum of the maker's fee for one account and the taker's fee for another account will always be non-negative. This lack of a global check on the pair of fee rates could lead to situations where the fee account might still lose funds. For example, trading when account 1 is maker and account 2 is taker has the `net_fee = -1%`.

```
Account 1: maker_fee = -1%, taker_fee = 2%
Account 2: maker_fee = 0%, taker_fee = 0%
```

Recommendation(s): Consider disallowing negative fee rates for both the maker and taker. If this is not feasible, implement additional logic in the `_fee_payment()` function to ensure that positive fees are always sufficient to cover any negative fees, preventing the fee account from losing funds.

Status: Fixed

Update from the client: Fixed in this PR #30.

7.3.5 Vault Is Never A Caller In Registry Contract

File(s): vaults/src/registry/registry.cairo

Description: The `Registry` contract enforces transfer restrictions between Paradex accounts. One restriction is that only the `Vault` contract can transfer assets from the operator to the `Vault` or auxiliary accounts. This is implemented by verifying the caller is the `Vault`, as shown in the snippet below:

```
if sender_is_operator
      && recipient_is_vault
      && self.operator_to_vault.read(sender) == recipient {
      let caller = get_caller_address();
      assert(self.vault_to_operator.read(caller) == sender, 'Caller is not the vault');
      return TransferRestriction::NoRestriction.into();
}
```

However, this implementation is flawed because the function `detect_transfer_restriction(...)`—which performs the restriction checks—is always invoked from the `Paraclear` exchange contract, not the `Vault` contract. As a result, the caller is the exchange contract, not the `Vault`, bypassing the intended restriction.

While transfer restrictions are not currently enforced for operator-initiated vault transfers, this implementation flaw would cause a denial of service for vault withdrawals if restrictions are implemented in the future.

Note: Furthermore, due to the previously reported issue about access control weaknesses in the `account_transfer_partial(...)` function, this issue was marked as medium based on the provided recommendations for that issue.

Recommendation(s): Implement a different mechanism to verify the true origin of the transfer request. For example, consider passing the intended caller's address as an additional parameter.

Status: Fixed

Update from the client: Fixed in this PR #7. Operators only transfer to vault on close, and this case is covered by `account_transfer_partial`. Extra check on the registry is to prevent any other transfer from the operator outside this workflow

7.3.6 Function `_transfer_positions_internal()` Fails To Create Asset Balance For Receiver If Asset Was Not Previously Used

File(s): paraclear/src/paraclear/paraclear.cairo

Description: The function `_transfer_positions_internal()` is designed to transfer part of a trader's position from one account to another. Prior to updating the position state for both the sender and receiver, it realizes any unrealized PNL and funding of the previous position, which are then reflected in the settlement token balance.

```
    let receiver_position = self
        .perpetual_future
        ._get_position_or_empty(receiver, market);
    let (
        receiver_updated_position,
        receiver_token_amount_realized,
        receiver_realized_funding_event,
        receiver_realized_pnl_event
    ) =
        self.perpetual_future.upsert_asset_balance(
            current_position: receiver_position,
            account: receiver,
            current_funding: funding_index,
            current_token_balance: updated_receiver_token_amount,
            order_side: liquidator_side,
            trade_request: @trade_request,
            settlement_token_asset_price: settlement_token_price
        );


    updated_receiver_token_amount = receiver_token_amount_realized;
    self.token.write_asset_balance(
            receiver,
            updated_receiver_token_amount,
            initial_receiver_token_balance
        );
```

However, this function uses `write_asset_balance()` to update the receiver's token balance without first verifying if the asset balance has been created for the receiver. The `get_asset_balance_or_empty()` function retrieves the `TokenAssetBalance` struct and assumes a new balance will be inserted at the end of the list if the asset has not been used previously.

In this scenario, the `token_address` is zero, and `get_asset_balance_or_empty()` inserts a new balance (through the returned values but without a storage update at this point), but it does not properly update the `Paraclear_token_asset_balance_tail`, which keeps track of the linked list of token balances. This results in the `Paraclear_token_asset_balance_tail` pointing to an incorrect tail, effectively breaking the linked list structure for the receiver's token balances.

Here is the relevant code for the `get_asset_balance_or_empty()` and `write_asset_balance()` functions:

```
    // token.cairo
    fn get_asset_balance_or_empty(
        self: @ComponentState<TContractState>,
        account: ContractAddress,
        token_address: ContractAddress
    ) -> TokenAssetBalance {
        let balance = self.Paraclear_token_asset_balance.read((account, token_address));
        if balance.token_address.is_zero() {
            let tail_token_address = self.Paraclear_token_asset_balance_tail.read(account);
            return TokenAssetBalance {
                token_address: token_address,
                amount: 0,
                prev: tail_token_address,
                next: Zero::zero(),
            };
        }
        balance
    }

    fn write_asset_balance(
        ref self: ComponentState<TContractState>,
        account: ContractAddress,
        amount: felt252,
        prev_balance: TokenAssetBalance
    ) {
        let balance = TokenAssetBalance {
            token_address: prev_balance.token_address,
            amount: amount,
            prev: prev_balance.prev,
            next: prev_balance.next,
        };
        self.Paraclear_token_asset_balance.write((account, balance.token_address), balance);
    }
```

Recommendation(s): Consider using a function `create_asset_balance()` to initialize the balance if it does not already exist, which will also update the `Paraclear_token_asset_balance_tail` correctly.

Status: Fixed

Update from the client: Fixed in this PR #46.

7.3.7 Function `is_risky()` Does Not Account For Trading Fee When Checking If The Trade Is Risky Or Not

File(s): paraclear/src/paraclear/paraclear.cairo

Description: In the current implementation of the `settleTrade()` function, the account states and balances of the maker and taker are loaded into temporary structs. Changes are applied to these structs during trade execution, and the `is_risky()` function is called with the updated account state to determine if the trade leaves the account with insufficient collateral to meet margin requirements.

```
let mut taker_state = self._load_account_with_market(taker_account, market);
// [...]
taker_state.replace_position_at(taker_state.trade_market_index, taker_updated_position);
taker_state
    .replace_token_at(
        taker_state.settlement_token_index,
        TokenAssetBalance {
            token_address: taker_initial_token_balance.token_address,
            amount: taker_updated_token_balance_d,
            prev: taker_initial_token_balance.prev,
            next: taker_initial_token_balance.next,
        }
    );

// Check if taker's position would be too risky
let taker_is_risky = taker_state
    .is_risky(taker_current_position.amount, taker_updated_position.amount);
// [...]
self
    ._fee_payments(
        maker_state,
        taker_state,
        trade.size.try_into().unwrap(),
        cost_price.try_into().unwrap(),
        @asset
    );
```

However, there is a critical flaw: the trading fee is not accounted for when calling the `is_risky()` function. The fee is applied afterward in the `_fee_payments()` function, meaning the temporary account state used in the `is_risky()` check does not reflect the reduced balances after the fee is deducted. This could allow the `is_risky()` check to pass incorrectly, even though the account becomes risky once the fees are applied.

Recommendation(s): Modify the implementation to apply trading fees to the temporary account state before invoking the `is_risky()` function. This ensures the risk assessment considers the post-fee account balances.

Status: Fixed

Update from the client: Fixed in this PR #14.

7.3.8 Closed Vault Cannot Transfer Assets to the Auxiliary Account

File(s): vaults/src/registry/registry.cairo

Description: This issue arises as a potential side effect of addressing a previously reported critical issue. It is assumed that transfer restrictions are enforced during the invocation of the `account_transfer_partial(...)` function.

When a user requests a withdrawal from the vault, assets are transferred from the `asset_holder` to the user's auxiliary account. Normally, the `asset_holder` is the vault operator. However, if the vault is closed, the `asset_holder` becomes the vault itself. In such cases, the registry's current transfer restrictions prevent the vault from transferring assets to the auxiliary account. Specifically, the registry contract disallows `Vault -> Auxiliary` transfers when the vault is closed, causing assets to remain stuck in the vault.

Recommendation(s): Update the registry's transfer restrictions to permit `Vault -> Auxiliary` transfers in cases where the vault is closed.

Status: Fixed

Update from the client: Fixed in this PR #7. Transfer from closed vault to auxiliary account is covered by `account_transfer_partial`. Extra check on the registry is to prevent direct transfer outside this workflow.

7.4 Low

7.4.1 Profit Shares Should Round Up

File(s): vaults/src/vault/vault.cairo

Description: When a user requests a withdrawal via the `request_withdrawal(...)` function, their deposited assets are compared to the current value of their shares. If the share value exceeds the deposited assets, a portion of the profit is shared with the vault owner, determined by the `profit_share_percentage` parameter.

The profit is calculated and converted to shares using the following code snippet:

```
// Calculate profit assets
let profit_assets = shares_value - withdraw_assets;
// Calculate profit share in assets
let profit_share_assets = (profit_assets * profit_share_percentage * CONSTANTS::WAD) / (100 * CONSTANTS::WAD);
// Convert profit share to shares
profit_share_shares = self._convert_to_shares(profit_share_assets);
```

The calculated `profit_share_shares` are transferred to the vault owner, who can redeem these shares later.

The issue arises because the `_convert_to_shares(...)` function rounds down the number of shares. Ideally, rounding should favor the vault owner to ensure accurate fee collection. The current implementation may allow users to bypass fees or pay less than intended in specific scenarios.

Recommendation(s): Modify the `_convert_to_shares(...)` function to round up when calculating the profit share in shares. This adjustment ensures that fees collected favor the vault owner.

Status: Fixed

Update from the client: Fixed in this PR #51. Added round up to profit shares.

7.4.2 `on_receive(...)` Does Not Create an Account

File(s): paraclear/src/paraclear/paraclear.cairo

Description: The `on_receive(...)` function is invoked by the bridge contract whenever a user bridges assets to the Paraclear contract. The bridge transfers funds directly to the Paraclear contract, and `on_receive(...)` updates the user's account balances accordingly.

However, this function does not invoke `self.account._add_new_account(...)`, which is required to create a new account for the user's address during their first deposit. This omission causes an inconsistency with the standard deposit functionality, where accounts are automatically created for first-time users.

Recommendation(s): Include a call to `self.account._add_new_account(...)` within the `on_receive(...)` function to ensure that a new account is created for users depositing funds for the first time via the bridge.

Status: Fixed

Update from the client: Fixed in this PR #45.

7.4.3 Users Could Add Duplicate Account Addresses to the Protocol

File(s): paraclear/src/account/account.cairo

Description: In the Account component, trader accounts are maintained in a linked list. The function `_add_new_account()` is responsible for appending new accounts to the tail of the list while attempting to prevent duplicate account addresses from being added. However, the current implementation contains a flaw.

The check to ensure that an account is not a duplicate relies on evaluating whether the `prev` or `next` links of the `current_account` are zero. This approach fails when the linked list contains only one account, as both `current_account.prev` and `current_account.next` are zero in this scenario. This allows the same account to bypass the check and be added multiple times, potentially creating a loop where an account's `prev` link points to itself.

```
fn _add_new_account(
    ref self: ComponentState<TContractState>, account_address: ContractAddress
) -> bool {
    // Only adds account if it doesn't already exist (prev and next are zero)
    let current_account = self.Paraclear_account.read(account_address);

    if current_account.prev != Zero::zero() || current_account.next != Zero::zero() {
        return true;
    }

    let current_tail = self.Paraclear_account_tail.read();
    let new_account = Account {
        account_address: account_address, prev: current_tail, next: Zero::zero(),
    };
    self.Paraclear_account.write(account_address, new_account);
    self.Paraclear_account_tail.write(account_address);
    if current_tail != Zero::zero() {
        let tail_account = self.Paraclear_account.read(current_tail);
        self
            .Paraclear_account
            .write(
                current_tail,
                Account {
                    account_address: current_tail,
                    prev: tail_account.prev,
                    next: account_address,
                }
            );
    }
    true
}
```

Recommendation(s): Update the implementation to directly verify the existence of the account using `current_account.account_address`. This approach ensures that the check is robust and eliminates the possibility of adding duplicate accounts.

Status: Fixed

Update from the Client: Fixed in this PR #45. Fixed check to be based on account address.

7.4.4 `taker_fee` or `maker_fee` Can Potentially Underflow the Balance

File(s): paraclear/src/paraclear/paraclear.cairo

Description: In the `settleTrade(...)` function, the maker and taker fees are calculated and deducted from user balances. The relevant code snippet is as follows:

```
let (maker_fee_d, maker_referrer, maker_fee_commission_d) = maker_state
    .get_trade_fee(trade_size, trade_price, true, asset);
let (taker_fee_d, taker_referrer, taker_fee_commission_d) = taker_state
    .get_trade_fee(trade_size, trade_price, false, asset);
let total_fee = maker_fee_d + taker_fee_d;
let maker_token_balance = *maker_state
    .token_balances[maker_state
    .settlement_token_index];
let taker_token_balance = *taker_state
    .token_balances[taker_state
    .settlement_token_index];

let maker_balance_after_fee = maker_token_balance.amount - maker_fee_d;
let taker_balance_after_fee = taker_token_balance.amount - taker_fee_d;
```

Since the fees are deducted directly from the user balances, there is a risk of underflow if the fee exceeds the available balance. If an underflow occurs (as the balance type is `felt252`), the user could receive a large token balance. This would not only allow unauthorized withdrawals but also cause severe accounting discrepancies within the protocol.

The severity of this issue depends on how fees are configured and whether they are kept below the user's margin requirement.

Recommendation(s): Introduce a check to ensure that the balance is greater than or equal to the fee before performing the deduction. For example:

```
if maker_token_balance.amount < maker_fee_d || taker_token_balance.amount < taker_fee_d {
    // Handle insufficient balance scenario
    return Err("Insufficient␣balance␣to␣cover␣fees");
}
```

This ensures that fees are only deducted when there are sufficient funds, preventing underflow and maintaining protocol integrity.

Status: Fixed

Update from the Client: Fixed in this PR #41.

7.4.5 Missing Validations in Function `settleTrade()`

File(s): paraclear/src/paraclear/paraclear.cairo

Description: The Paradex protocol integrates an offchain orderbook with an onchain smart contract for handling matched trades. The `settleTrade()` function processes these trades on-chain. However, it lacks several critical validations that ensure the integrity of the trading process:

- **Order Side Mismatch:** The function does not validate that the `side` fields of the taker and maker orders are opposite. For a valid trade, one order must be a buy and the other a sell. Without this check, the function could erroneously attempt to settle invalid trades between two buy or two sell orders.
- **Exceeding Order Size:** The function does not verify whether the total traded amount across multiple trades of the same order exceeds the order size. This could lead to settling trades that surpass the intended limit of the order.
- **Stale Orders:** The `signature_timestamp` field, although defined, is not utilized. This field could be used to ensure that orders have not expired before being settled, preventing stale or outdated orders from being processed.

Recommendation(s): To address these issues and enhance the security and reliability of the protocol, implement the following validations:

1. Ensure the `side` fields of the taker and maker orders are opposite (e.g., taker = buy, maker = sell) before proceeding with settlement.

```
    if taker_order.side == maker_order.side {
        return Err("Invalid trade: Both orders have the same side");
    }

```

2. Track the cumulative traded amount for each order and validate that it does not exceed the order size.

```
    if total_traded_amount > order.size {
        return Err("Invalid trade: Cumulative trade size exceeds order size");
    }

```

3. Implement logic to verify the `signature_timestamp` field and ensure that orders are not expired.

```
    if signature_timestamp < block_timestamp {
        return Err("Invalid trade: Order has expired");
    }

```

These measures would prevent invalid trades, stale orders, and incorrect cumulative trades, ensuring that the protocol adheres to expected standards.

Status: Fixed

Update from the Client: Fixed in this PR #50 and this PR #52.

7.5 Informational

7.5.1 The Function Name Does Not Reflect The Returned Data

File(s): paraclear/src/perpetual/future.cairo

Description: The function `get_total_unrealized_funding_pnl(...)` is intended to return the total unrealized funding for a specific futures market. However, its implementation retrieves data from `Paraclear_total_perpetual_asset_realized_funding_pnl`, which tracks realized funding instead of unrealized funding. This creates a mismatch between the function name and its actual behavior, leading to potential confusion for developers.

```
fn get_total_unrealized_funding_pnl(
        self: @ComponentState<TContractState>, market: felt252
    ) -> felt252 {
        let pnl = self.Paraclear_total_perpetual_asset_realized_funding_pnl.read(market);
        return pnl;
}
```

Recommendation(s): To resolve this issue:

1. **Rename the function**: Update the function name to reflect that it retrieves realized funding, such as `get_total_realized_funding_pnl(...)`.
2. **Update the logic**: If the function is intended to retrieve unrealized funding data, modify the logic to fetch the correct data from the appropriate source.

These updates will ensure consistency between the function's name and its actual behavior, enhancing code clarity and maintainability.

Status: Fixed

Update from the client: Fixed in this PR #42.

7.5.2 Sub-operator Cannot Be Registered

File(s): vaults/src/registry/registry.cairo

Description: The `Registry` contract manages the registration of operators, vaults, and auxiliary accounts. It also provides functionality for registering sub-operators through the `register_sub_operator(...)` function. This function is restricted to being called exclusively by the `Factory` contract:

```
fn register_sub_operator(
    ref self: ContractState,
    operator: ContractAddress,
    sub_operator: ContractAddress
) {
    self.assert_caller_is_factory();
    self.sub_operator_to_operator.write(sub_operator, operator);
    self.emit(SubOperatorRegistered { operator, sub_operator });
}
```

However, the `Factory` contract does not implement any function capable of invoking `register_sub_operator(...)`. As a result, this functionality remains completely unreachable, rendering the `register_sub_operator(...)` method ineffective.

Recommendation(s): Add a public function in the `Factory` contract that can invoke the `register_sub_operator(...)` function within the `Registry` contract.

Status: Fixed

Update from the client: Fixed in this PR #6 and this PR #10

7.5.3 Disallow Withdrawals Which Are Below Scaling Factor

File(s): vaults/src/paraclear/paraclear.cairo

Description: During withdrawals, the requested amount is divided by a scaling factor:

```
let amount: felt252 = mul_128(
    requested_amount.try_into().unwrap(), ONE - socialized_loss_factor
)
    .try_into().unwrap();
// ...
let amount_u256: u256 = amount.into();
let amount_scaled: u256 = amount_u256 / self._scale_factor(decimals);
// ...
let is_transfer_successful = token_dispatcher.transfer(recipient, amount_scaled);
// ...
let updated_balance = self
    .upsert_asset_balance(recipient, token_address, -requested_amount);
// ...
```

The `_scale_factor(...)` function checks whether the token being withdrawn has six decimals, and if so, it returns 100. Consequently, the transferred amount is divided by 100, but the originally requested amount is deducted from the user's balance. This mismatch causes users to lose one wei of assets if they withdraw an amount less than 100.

Recommendation(s): Disallow withdrawals where the requested amount is less than the scaling factor.

Status: Fixed

Update from the client: Fixed in this PR #43. We only disallowed withdrawals with zero amount, now dust is transferred to the insurance fund.

7.5.4 `liquidate` Should Calculate Penalty Settlement Value Too

File(s): paraclear/src/paraclear/paraclear.cairo

Description: During partial liquidation, the USDC value of the liquidation penalty is calculated before being passed to the `transfer_internal(...)` function:

```
let liq_penalty_full_value = div_128(
    liq_penalty_full, settlement_token_price.try_into().unwrap()
);
let liq_penalty = mul_128(
    liq_penalty_full_value, liquidation_share.try_into().unwrap()
);
```

However, this calculation is not performed within the `liquidate(...)` function:

```
let margin_requirement: i128 = account_state
  .margin_requirement(MARGIN_CHECK_MAINTENANCE)
  .try_into()
  .unwrap();
let liq_penalty_d = mul_128(margin_requirement, liquidation_fee.try_into().unwrap())
  .into();
self._transfer_positions_internal(account_state, liquidator, 0, 1, true);
self.token.liquidate_full(
        account,
      liquidator,
      insurance_account,
      self.getSettlementTokenAsset(),
      liq_penalty_d);
```

As a result, the liquidation penalty may be inconsistent with the collateral balance. This is because the penalty is calculated in USD value but deducted from collateral in USDC, leading to potential inaccuracies.

Recommendation(s): Calculate the USDC value of the liquidation penalty directly within the `liquidate(...)` function to ensure consistency with the collateral balance.

Status: Fixed

Update from the client: Removed legacy full liquidation logic, penalty is only calculated in case of partial liquidations.

7.5.5 `settle_market(...)` Can Potentially Lead To Account Bankruptcy

File(s): paraclear/src/paraclear/paraclear.cairo

Description: The `settle_market(...)` function is responsible for closing open positions and realizing the current profits and losses, which are immediately reflected in the account's balance. Since this function is callable only by the executor, the off-chain code of the executor is solely responsible for validating the settlement and executing the appropriate transactions.

However, there is a potential risk where the settlement could bankrupt an account if the off-chain code allows `settle_market` to be executed when the account should instead be liquidated.

Recommendation(s): Introduce a check within the `settle_market(...)` function to prevent its execution when the account meets conditions that warrant liquidation.

Status: Acknowledged

Update from the client: Prevent market settling would introduce the risk of keeping positions open for markets no longer valid and increase the complexity and the timeliness of the process. If account is unhealthy the insurance fund will take what remains of the account in any case. The difference is that on settle market the position will be unwound on the user account, in case of liquidations it will be on the insurance fund.

7.5.6 `liquidate_partial(...)` Could Check If The Partial Liquidation Made An Account Healthy

File(s): paraclear/src/paraclear/paraclear.cairo

Description: The `liquidate_partial(...)` function is designed to partially liquidate a specific account that is not sufficiently healthy. Liquidation is performed based on the `liquidation_share` value provided as input by the executor.

Although the executor's input is trusted, it requires the executor to calculate off-chain the exact number of shares necessary to restore the account's health. However, the function lacks any on-chain validation to ensure the account's health after the partial liquidation is executed.

Recommendation(s): Implement an additional check within the function to verify that the account is healthy following a partial liquidation.

Status: Acknowledged

Update from the client: This is the intended behavior: in the case account will remain unhealthy (could potentially happens if prices move while liquidation is inflight) it will partially liquidated and just after a new partial liquidation will be executed to finalize the remaining portion.

7.5.7 Incorrect Value Passed To `is_liquidation` When Calling `transfer_internal()` In The `account_transfer_partial()` Function

File(s): paraclear/src/paraclear/paraclear.cairo

Description: In the token component, the function `transfer_internal()` has an input parameter `is_liquidation`, which is intended to indicate that the transfer is part of the liquidation process. When `is_liquidation` is set to 1, the function will emit an event to signal that the transfer occurred within liquidation.

However, in the `account_transfer_partial()` function, the `is_liquidation` parameter is incorrectly set to 1, even though the transfer is not related to liquidation. This results in the potential for incorrect event emissions, as the function is not intended for liquidation transfers.

```
fn account_transfer_partial(
    ref self: ContractState,
    account: ContractAddress,
    receiver: ContractAddress,
    account_share: felt252
) -> felt252 {
    // [..]
    self
        .token
        .transfer_internal(
            account, receiver, self.getSettlementTokenAsset(), token_transfer.into(), 1
        );
    // [..]
}
```

Recommendation(s): Consider updating the value passed to the `is_liquidation` parameter in the `account_transfer_partial()` function to ensure correct event emission.

Status: Acknowledged

Update from the client: Name on both function parameter and event field are misleading, but the behavior is that intended. It is used to communicate to cloud component that balance update is not coming from a trade. We will consider to rename them future, but since the current behavior is correct the operational effort to synchronize off-chain components is not worth doing for now.

7.5.8 SRC5 External Function Not Exposed

File(s): vaults/src/vault/vault.cairo, vaults/src/registry/registry.cairo

Description: The SRC5 interface is utilized in the `vault.cairo` and `registry.cairo` contracts, but the `supports_interface()` function is not exposed externally. While the SRC5 component is being used, the necessary implementation to expose the `supports_interface()` function via the `SRC5Impl` is missing in both contracts.

```
// Components
component!(path: AccessControlComponent, storage: accesscontrol, event: AccessControlEvent);
component!(path: SRC5Component, storage: src5, event: SRC5Event);
component!(path: UpgradeableComponent, storage: upgradeable, event: UpgradeableEvent);

impl InternalUpgradeableImpl = UpgradeableComponent::InternalImpl<ContractState>;

#[abi(embed_v0)]
impl AccessControlImpl =
    AccessControlComponent::AccessControlImpl<ContractState>;
impl AccessControlInternalImpl = AccessControlComponent::InternalImpl<ContractState>;
```

Recommendation(s): Consider implementing the `SRC5Impl` to expose the `supports_interface()` function in both contracts.

Status: Fixed

Update from the client: Fixed in this PR #53.

7.5.9 Unused Code

File(s): *.cairo

Description: There are instances of the code which are not used and do not contain the TODO option:

- Four out of six fields in the `TokenAsset` struct are not used:

```
// @audit Fields initial_weight, maintenance_weight, conversion_weight, tick_size are never used
pub struct TokenAsset {
    pub initial_weight: felt252,
    pub maintenance_weight: felt252,
    pub conversion_weight: felt252,
    pub tick_size: felt252,
    pub token_address: ContractAddress,
    pub token_name: felt252,
}
```

- In the storage of `future.cairo` two storage variables are not used:

```
  pub Paraclear_perpetual_asset_link_tail: felt252`
  pub Paraclear_perpetual_asset_link: Map::<felt252, PerpetualAssetLink>
```

- These functions are not used anywhere in the code:

  - `assert_caller_is_vault_operator(...)` in `vault.cairo`
  - `append_position(...)` in `account.cairo`
  - `_assert_only_state_publisher` in `paraclear.cairo`

Recommendation(s): Consider removing unused functions and storage variables. If these variables and functions are meant to be used later, ensure they are properly initialized and integrated into the relevant code.

Status: Fixed

Update from the client: Acknowledged on `TokenAsset` fields, for now we can keep them for future use;

  - asset link storage removed in PR #33
  - `append_position` removed in PR #10
  - `_assert_only_state_publisher` is now in use (see PR #13)
  - `assert_caller_is_vault_operator(...)` removed in PR #49

7.6 Best Practices

7.6.1 Percentages Should Have Upper Bound

File(s): vaults/src/factory/factory.cairo

Description: The `Factory` contract is responsible for deploying vaults and configuring their initial parameters. Several percentage parameters are set during deployment but lack upper bounds, which could lead to misconfiguration. These parameters are set by the following functions:

- `set_max_profit_share_percentage(...)`: Configures the maximum percentage for profit sharing with the vault owner.
- `set_min_owner_share_percentage(...)`: Defines the minimum percentage of vault shares the owner is required to hold.

While these parameters can only be set for future deployments, it is a best practice to enforce upper bounds for percentage values. Without limits, values could exceed reasonable thresholds. Establishing a reasonable range ensures the parameters remain within acceptable limits.

Recommendation(s): Consider introducing upper bounds for the percentage storage variables in the `Factory` contract.

Status: Fixed

Update from the client: Fixed in this PR #44.

7.6.2 Unnecessary Multiplication with WAD in Both Numerator and Denominator

File(s): vaults/src/vault/vault.cairo

Description: In the function `request_withdrawal()`, the value of `profit_share_assets` is computed by multiplying `profit_assets` with `profit_share_percentage`, then dividing by `100`. However, the multiplication of `CONSTANTS::WAD` in both the numerator and denominator is unnecessary and does not affect the result.

```
if (shares_value > withdraw_assets && profit_share_percentage > 0 && !is_owner) {
    // profit of the account
    let profit_assets = shares_value - withdraw_assets;
    // profit share in assets

    let profit_share_assets = (profit_assets * profit_share_percentage * CONSTANTS::WAD)
        / (100 * CONSTANTS::WAD);
    // profit share in shares
    profit_share_shares = self._convert_to_shares(profit_share_assets);
    // transfer profit share to the owner from vault (transferred to vault before)
    self.erc20._transfer(caller, owner, profit_share_shares);
}
```

Recommendation(s): Remove the multiplication by `CONSTANTS::WAD` from both the numerator and the denominator as it does not alter the final result and simplifies the calculation.

Status: Fixed

Update from the client: Fixed in this PR #49.

8 Post Audit Recommendations

Our review has highlighted several steps that would improve the Paradex codebase and enhance its reliability. We recommend the following actions:

- **Completing Development of Unfinished Features:** There are incomplete features and unresolved TODO items in the codebase. Addressing these will help ensure the system is fully functional and consistent.
- **Improving Test Coverage:** Expanding and refining the test suites will provide better coverage and help identify potential issues earlier, improving the overall reliability of the system.
- **Carefully Implementing Fixes:** Some of the identified issues could have a significant impact on the overall codebase. Therefore, we recommend that fixes be implemented with careful planning and testing to prevent unintended consequences. Proper testing practices should also be adopted as part of this process to ensure that similar issues are avoided in the future.
- **Performing a Follow-Up Audit:** After these improvements are made, a second audit is strongly recommended to identify any remaining issues and further refine the codebase.

By following these recommendations, the Paradex codebase will become more robust, reliable, and better prepared for future use.

9 Test Evaluation

9.1 Compilation Output

```
scarb build
   Updating git repository https://github.com/openzeppelin/cairo-contracts
Downloading alexandria_data_structures v0.5.0
Downloading snforge_scarb_plugin v0.41.0
Downloading snforge_std v0.41.0
  Compiling paradex v1.0.0 (/home/boreas/paradex/Scarb.toml)
   Finished `dev` profile target(s) in 36 seconds
```

9.2 Tests Output

```
snforge test --workspace

   Compiling snforge_scarb_plugin v0.41.0
    Finished `release` profile [optimized] target(s) in 0.21s
   Compiling test(paradex_unittest) paradex v1.0.0 (/home/boreas/paradex/Scarb.toml)
   Compiling test(paradex_oracle_unittest) paradex_oracle v1.0.0 (/home/boreas/paradex/oracle/Scarb.toml)
   Compiling test(paradex_paraclear_unittest) paradex_paraclear v1.0.0 (/home/boreas/paradex/paraclear/Scarb.toml
    )
   Compiling test(paradex_registry_unittest) paradex_registry v1.0.0 (/home/boreas/paradex/registry/Scarb.toml)
warn: external contracts not found for selectors: `paradex_paraclear::paraclear::Paraclear`
   Compiling test(paradex_test_common) paradex_test_common v1.0.0 (/home/boreas/paradex/test_common/Scarb.toml)
   Compiling test(paradex_vaults_unittest) paradex_vaults v1.0.0 (/home/boreas/paradex/vaults/Scarb.toml)
warn: external contracts not found for selectors: `paradex_paraclear::paraclear::Paraclear`
    Finished `dev` profile target(s) in 7 minutes


Collected 0 test(s) from paradex package
Running 0 test(s) from src/
Tests: 0 passed, 0 failed, 0 skipped, 0 ignored, 0 filtered out


Collected 4 test(s) from paradex_oracle package
Running 4 test(s) from tests/
[PASS] paradex_oracle::tests::test_oracle::upgrade_oracle_wrong_caller (l1_gas: ~0, l1_data_gas: ~384, l2_gas:
    ~1451200)
[PASS] paradex_oracle::tests::test_oracle::upgrade_oracle_zero_class_hash (l1_gas: ~0, l1_data_gas: ~384, l2_gas:
     ~1651200)
[PASS] paradex_oracle::tests::test_oracle::upgrade_oracle (l1_gas: ~0, l1_data_gas: ~384, l2_gas: ~2106560)
[PASS] paradex_oracle::tests::test_oracle::test_get_value (l1_gas: ~0, l1_data_gas: ~768, l2_gas: ~2491200)
Tests: 4 passed, 0 failed, 0 skipped, 0 ignored, 0 filtered out


Collected 204 test(s) from paradex_paraclear package
Running 204 test(s) from tests/
[IGNORE] paradex_paraclear::token::tests::test_token::test_create_token_asset
[IGNORE] paradex_paraclear::paraclear::tests::test_paraclear_trades::test_settle_trade_many_positions
[PASS] paradex_paraclear::account::tests::test_account::test_add_account (l1_gas: ~0, l1_data_gas: ~1344, l2_gas:
     ~5019200)
[PASS] paradex_paraclear::account::tests::test_account::test_add_account_alias (l1_gas: ~0, l1_data_gas: ~1344,
    l2_gas: ~5019200)
[PASS] paradex_paraclear::account::tests::test_account::test_account_state (l1_gas: ~0, l1_data_gas: ~1152,
    l2_gas: ~4139200)
[PASS] paradex_paraclear::account::tests::test_account::test_add_account_configurator_unauthorized (l1_gas: ~0,
    l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::account::tests::test_account::test_add_account_executor_unauthorized (l1_gas: ~0,
    l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::account::tests::test_account::test_add_account_no_role_unauthorized (l1_gas: ~0,
    l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::account::tests::test_account::test_add_account_state_publisher_unauthorized (l1_gas:
    ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::account::tests::test_account::test_all_account_states (l1_gas: ~0, l1_data_gas: ~1152,
    l2_gas: ~4139200)
[PASS] paradex_paraclear::account::tests::test_account::test_all_account_states_alias (l1_gas: ~0, l1_data_gas:
    ~1152, l2_gas: ~4139200)
```

```
[PASS] paradex_paraclear::account::tests::test_account::test_get_account_fee_rate (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4259200)
[PASS] paradex_paraclear::account::tests::test_account::test_get_account_referral (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4219200)
[PASS] paradex_paraclear::account::tests::test_account::test_get_accounts (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4139200)
[PASS] paradex_paraclear::account::tests::test_account::test_remove_account (l1_gas: ~0, l1_data_gas: ~1344, l2_gas: ~5099200)
[PASS] paradex_paraclear::account::tests::test_account::test_get_account_state (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4179200)
[PASS] paradex_paraclear::account::tests::test_account::test_get_accounts_alias (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4139200)
[PASS] paradex_paraclear::account::tests::test_account::test_remove_account_configurator_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::account::tests::test_account::test_remove_account_alias (l1_gas: ~0, l1_data_gas: ~1344, l2_gas: ~5099200)
[PASS] paradex_paraclear::account::tests::test_account::test_remove_account_executor_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::account::tests::test_account::test_remove_account_state_publisher_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::account::tests::test_account::test_set_account_fee_rate (l1_gas: ~0, l1_data_gas: ~1440, l2_gas: ~4979200)
[PASS] paradex_paraclear::account::tests::test_account::test_remove_account_no_role_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::account::tests::test_account::test_set_account_fee_rate_executor_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::account::tests::test_account::test_set_account_fee_rate_state_publisher_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::account::tests::test_account::test_set_account_fee_rate_no_role_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::account::tests::test_account::test_set_account_referral_alias (l1_gas: ~0, l1_data_gas: ~1440, l2_gas: ~5289920)
[PASS] paradex_paraclear::account::tests::test_account::test_set_account_referral (l1_gas: ~0, l1_data_gas: ~1440, l2_gas: ~5289920)
[PASS] paradex_paraclear::account::tests::test_account::test_set_account_referral_alias_executor_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_math::test_abs_128 (l1_gas: ~0, l1_data_gas: ~0, l2_gas: ~80000)
[PASS] paradex_paraclear::paraclear::tests::test_math::test_div_128 (l1_gas: ~0, l1_data_gas: ~0, l2_gas: ~720000)
[PASS] paradex_paraclear::account::tests::test_account::test_set_global_fee_rate_negative_fee (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::account::tests::test_account::test_set_global_fee_rate_no_role_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_math::test_max_128 (l1_gas: ~0, l1_data_gas: ~0, l2_gas: ~80000)
[PASS] paradex_paraclear::account::tests::test_account::test_set_global_fee_rate_state_publisher_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_math::test_min_128 (l1_gas: ~0, l1_data_gas: ~0, l2_gas: ~80000)
[PASS] paradex_paraclear::paraclear::tests::test_math::test_mul3_128 (l1_gas: ~0, l1_data_gas: ~0, l2_gas: ~1960000)
[PASS] paradex_paraclear::paraclear::tests::test_math::test_mul_128 (l1_gas: ~0, l1_data_gas: ~0, l2_gas: ~880000)
[PASS] paradex_paraclear::paraclear::tests::test_math::test_round_down_128 (l1_gas: ~0, l1_data_gas: ~0, l2_gas: ~240000)
[PASS] paradex_paraclear::account::tests::test_account::test_set_account_referral_alias_no_role_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_math::test_round_up_128 (l1_gas: ~0, l1_data_gas: ~0, l2_gas: ~280000)
[PASS] paradex_paraclear::account::tests::test_account::test_set_account_referral_alias_state_publisher_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::account::tests::test_account::test_set_account_referral_executor_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::account::tests::test_account::test_set_account_referral_no_role_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::account::tests::test_account::test_set_account_referral_state_publisher_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_delegated_market (l1_gas: ~0, l1_data_gas: ~2016, l2_gas: ~6970880)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_decimals (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4139200)
```

```
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_get_account_free_balance (l1_gas: ~0, l1_data_gas: ~2016, l2_gas: ~8610880)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_delegate_market (l1_gas: ~0, l1_data_gas: ~2112, l2_gas: ~7650880)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_delegate_market_executor_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_delegate_market_no_role_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_delegate_market_state_publisher_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_get_settlement_token_asset (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4179200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_get_total_token_balance (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4139200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_get_total_token_tvl (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4139200)
[PASS] paradex_paraclear::account::tests::test_account::test_set_global_fee_rate (l1_gas: ~0, l1_data_gas: ~1440, l2_gas: ~4939200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_get_version (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4139200)
[PASS] paradex_paraclear::account::tests::test_account::test_set_global_fee_rate_executor_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::
    test_set_settlement_token_asset_configurator_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_set_settlement_token_asset_executor_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_set_settlement_token_asset_no_role_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_get_account_free_balance_alias (l1_gas: ~0, l1_data_gas: ~2016, l2_gas: ~8610880)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_get_account_health_check (l1_gas: ~0, l1_data_gas: ~2016, l2_gas: ~8610880)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_get_account_liquidation_penalty (l1_gas: ~0, l1_data_gas: ~2016, l2_gas: ~8530880)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_get_account_margin_fraction (l1_gas: ~0, l1_data_gas: ~2016, l2_gas: ~8490880)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_liquidations::
    test_set_insurance_fund_account_no_role_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_settle_market_alias_configurator_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::
    test_set_settlement_token_asset_state_publisher_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_settle_market_alias_no_role_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_settle_market_alias_state_publisher_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_liquidations::
    test_set_insurance_fund_account_state_publisher_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_settle_market_configurator_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_liquidations::test_set_liquidation_fee (l1_gas: ~0, l1_data_gas: ~1248, l2_gas: ~4939200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_liquidations::
    test_set_liquidation_fee_executor_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_liquidations::
    test_set_liquidation_fee_no_role_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_trades::test_set_fee_account_no_role_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_invariants_check (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4139200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_invariants_check_by_market (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4139200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_settle_market_no_role_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_trades::
    test_set_fee_account_state_publisher_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_settle_market_state_publisher_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
```

```
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_upgrade_executor_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4379200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_upgrade_configurator_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4379200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_get_account_margin_requirement_by_market (l1_gas: ~0, l1_data_gas: ~2016, l2_gas: ~8490880)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_get_account_state_alias (l1_gas: ~0, l1_data_gas: ~2016, l2_gas: ~8570880)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_liquidations::test_set_liquidation_fee_state_publisher_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_get_account_unrealized_funding_pnl (l1_gas: ~0, l1_data_gas: ~2016, l2_gas: ~8490880)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_trades::test_get_fee_account (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4179200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_trades::test_get_fee_share_account (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4179200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_trades::test_get_fee_share_percentage (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4139200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_trades::test_set_fee_account (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4979200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_set_bridge_contract (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4699200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_set_bridge_contract_configurator_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_set_bridge_contract_no_role_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_set_bridge_contract_executor_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_trades::test_settle_trade_same_side_orders_not_allowed_buy (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4379200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_upgrade_no_role_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4379200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_upgrade_state_publisher_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4379200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_trades::test_settle_trade_same_side_orders_not_allowed_sell (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4379200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_trades::test_settle_trade_size_cannot_be_one (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4379200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_trades::test_settle_trade_self_trade_not_allowed (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4379200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_transfers::test_transfer (l1_gas: ~0, l1_data_gas: ~2016, l2_gas: ~6810880)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_transfers::test_withdraw (l1_gas: ~0, l1_data_gas: ~2016, l2_gas: ~6810880)
[PASS] paradex_paraclear::perpetual::tests::test_perp_future::test_create_perpetual_asset (l1_gas: ~0, l1_data_gas: ~1920, l2_gas: ~5390400)
[PASS] paradex_paraclear::perpetual::tests::test_perp_future::test_create_perpetual_asset_executor_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::perpetual::tests::test_perp_future::test_create_perpetual_asset_fails_with_duplicate_market (l1_gas: ~0, l1_data_gas: ~1920, l2_gas: ~4990400)
[PASS] paradex_paraclear::perpetual::tests::test_perp_future::test_create_perpetual_asset_no_role_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::perpetual::tests::test_perp_future::test_create_perpetual_asset_fails_with_zero_market (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::perpetual::tests::test_perp_future::test_create_perpetual_asset_state_publisher_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::perpetual::tests::test_perp_future::test_get_perpetual_asset (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::perpetual::tests::test_perp_future::test_get_perpetual_futures_mmf_factor (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4179200)
[PASS] paradex_paraclear::perpetual::tests::test_perp_future::test_update_perpetual_asset (l1_gas: ~0, l1_data_gas: ~1920, l2_gas: ~6041600)
[PASS] paradex_paraclear::perpetual::tests::test_perp_future::test_update_perpetual_asset_fails_with_nonexistent_market (l1_gas: ~0, l1_data_gas: ~1920, l2_gas: ~4990400)
[PASS] paradex_paraclear::perpetual::tests::test_perp_future::test_update_perpetual_asset_fails_with_different_base_asset (l1_gas: ~0, l1_data_gas: ~1920, l2_gas: ~4990400)
[PASS] paradex_paraclear::perpetual::tests::test_perp_future::test_update_perpetual_asset_fails_with_zero_market (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
```

```
[PASS] paradex_paraclear::perpetual::tests::test_perp_future::test_update_perpetual_futures_mmf_factor_executor_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::perpetual::tests::test_perp_future::test_update_perpetual_futures_mmf_factor_max_value (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::perpetual::tests::test_perp_future::test_update_perpetual_futures_mmf_factor (l1_gas: ~0, l1_data_gas: ~1248, l2_gas: ~6460160)
[PASS] paradex_paraclear::perpetual::tests::test_perp_future::test_update_perpetual_futures_mmf_factor_no_role_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::perpetual::tests::test_perp_future::test_update_perpetual_futures_mmf_factor_state_publisher_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::perpetual::tests::test_perp_option::test_create_perpetual_option_asset_cannot_create_twice (l1_gas: ~0, l1_data_gas: ~2688, l2_gas: ~5886720)
[PASS] paradex_paraclear::perpetual::tests::test_perp_option::test_create_perpetual_option_asset_call (l1_gas: ~0, l1_data_gas: ~2688, l2_gas: ~6246720)
[PASS] paradex_paraclear::perpetual::tests::test_perp_option::test_create_perpetual_option_asset_executor_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::perpetual::tests::test_perp_future_balance::test_create_and_remove_asset_balance (l1_gas: ~0, l1_data_gas: ~2784, l2_gas: ~28976000)
[PASS] paradex_paraclear::perpetual::tests::test_perp_future_balance::test_asset_balance_linked_list (l1_gas: ~0, l1_data_gas: ~4128, l2_gas: ~42604160)
[PASS] paradex_paraclear::perpetual::tests::test_perp_future_balance::test_funding_realization (l1_gas: ~0, l1_data_gas: ~3360, l2_gas: ~27856000)
[PASS] paradex_paraclear::perpetual::tests::test_perp_option::test_create_perpetual_option_asset_no_role_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::perpetual::tests::test_perp_option::test_create_perpetual_option_asset_put (l1_gas: ~0, l1_data_gas: ~2688, l2_gas: ~6246720)
[PASS] paradex_paraclear::perpetual::tests::test_perp_option::test_create_perpetual_option_asset_state_publisher_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::perpetual::tests::test_perp_option::test_create_perpetual_option_asset_zero_market (l1_gas: ~0, l1_data_gas: ~2112, l2_gas: ~5125760)
[PASS] paradex_paraclear::perpetual::tests::test_perp_option::test_create_perpetual_option_asset_without_cross_margin_parameters (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::perpetual::tests::test_perp_option::test_create_perpetual_option_asset_zero_market_invalid_option_type (l1_gas: ~0, l1_data_gas: ~2112, l2_gas: ~5125760)
[PASS] paradex_paraclear::perpetual::tests::test_perp_option::test_create_perpetual_option_margin_params (l1_gas: ~0, l1_data_gas: ~2112, l2_gas: ~5605760)
[PASS] paradex_paraclear::perpetual::tests::test_perp_option::test_create_perpetual_option_margin_params_cannot_create_twice (l1_gas: ~0, l1_data_gas: ~2112, l2_gas: ~5125760)
[PASS] paradex_paraclear::perpetual::tests::test_perp_option::test_create_perpetual_option_margin_params_executor_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::perpetual::tests::test_perp_option::test_create_perpetual_option_margin_params_no_role_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::perpetual::tests::test_perp_option::test_create_perpetual_option_margin_params_state_publisher_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::perpetual::tests::test_perp_option::test_get_perpetual_option_asset (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4259200)
[PASS] paradex_paraclear::perpetual::tests::test_perp_option::test_get_perpetual_option_margin_params (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4379200)
[PASS] paradex_paraclear::perpetual::tests::test_perp_option::test_set_perpetual_options_fee_cap (l1_gas: ~0, l1_data_gas: ~1344, l2_gas: ~4959680)
[PASS] paradex_paraclear::perpetual::tests::test_perp_option::test_get_perpetual_options_fee_cap (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4219200)
[PASS] paradex_paraclear::perpetual::tests::test_perp_option::test_set_perpetual_options_fee_cap_executor_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::perpetual::tests::test_perp_option::test_set_perpetual_options_fee_cap_no_role_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::perpetual::tests::test_perp_option::test_set_perpetual_options_fee_cap_state_publisher_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
```

```
[PASS] paradex_paraclear::perpetual::tests::test_perp_option::test_update_perpetual_option_asset_cannot_update_non_existing_assets (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::perpetual::tests::test_perp_option::test_update_perpetual_option_asset (l1_gas: ~0, l1_data_gas: ~2688, l2_gas: ~7007680)
[PASS] paradex_paraclear::perpetual::tests::test_perp_option::test_update_perpetual_option_asset_change_base_asset (l1_gas: ~0, l1_data_gas: ~3648, l2_gas: ~6673280)
[PASS] paradex_paraclear::perpetual::tests::test_perp_option::test_update_perpetual_option_asset_change_option_type (l1_gas: ~0, l1_data_gas: ~2688, l2_gas: ~7007680)
[PASS] paradex_paraclear::perpetual::tests::test_perp_option::test_update_perpetual_option_asset_change_quote_asset (l1_gas: ~0, l1_data_gas: ~2688, l2_gas: ~7007680)
[PASS] paradex_paraclear::perpetual::tests::test_perp_option::test_update_perpetual_option_asset_executor_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::perpetual::tests::test_perp_option::test_update_perpetual_option_asset_change_strike (l1_gas: ~0, l1_data_gas: ~2688, l2_gas: ~7007680)
[PASS] paradex_paraclear::perpetual::tests::test_perp_option::test_update_perpetual_option_asset_change_tick_size (l1_gas: ~0, l1_data_gas: ~2688, l2_gas: ~7007680)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_get_account_unrealized_pnl (l1_gas: ~0, l1_data_gas: ~2016, l2_gas: ~8490880)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_get_account_unrealized_pnl_by_market (l1_gas: ~0, l1_data_gas: ~2016, l2_gas: ~7690880)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_get_name (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4139200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_trades::test_settle_trade_size_cannot_be_zero (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4379200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_get_account_value (l1_gas: ~0, l1_data_gas: ~2016, l2_gas: ~8570880)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_trades::test_settle_trade_state_publisher_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4379200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_trades::test_settle_trade_timetamp_cannot_be_zero (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4379200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_transfers::test_account_transfer_partial_max_share (l1_gas: ~0, l1_data_gas: ~2016, l2_gas: ~6930880)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_trades::test_set_fee_account_configurator_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::perpetual::tests::test_perp_option::test_update_perpetual_option_asset_without_cross_margin_parameters (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::perpetual::tests::test_perp_option::test_update_perpetual_option_margin_params (l1_gas: ~0, l1_data_gas: ~2112, l2_gas: ~6392320)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_set_bridge_contract_state_publisher_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_trades::test_set_fee_account_executor_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_set_oracle_contract (l1_gas: ~0, l1_data_gas: ~2016, l2_gas: ~6970880)
[PASS] paradex_paraclear::perpetual::tests::test_perp_option::test_update_perpetual_option_margin_params_executor_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_set_oracle_contract_configurator_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::perpetual::tests::test_perp_option::test_update_perpetual_option_asset_no_role_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::perpetual::tests::test_perp_option::test_update_perpetual_option_asset_state_publisher_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::perpetual::tests::test_perp_option::test_update_perpetual_option_margin_params_state_publisher_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::perpetual::tests::test_perp_option::test_update_perpetual_option_margin_params_no_role_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::upgrade_paraclear (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4834560)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::upgrade_paraclear_zero_class_hash (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_liquidations::test_account_liquidate_configurator_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_liquidations::test_account_liquidate (l1_gas: ~0, l1_data_gas: ~2016, l2_gas: ~9796480)
```

```
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_trades::test_set_fee_share_cannot_be_greater_than_90 (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_liquidations::test_account_liquidate_invalid_liquidator (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~5684800)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_get_settlement_asset_total_balance (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4219200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_get_oracle_contract (l1_gas: ~0, l1_data_gas: ~2016, l2_gas: ~6970880)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_trades::test_set_fee_share_configurator_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_trades::test_set_fee_share_state_publisher_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_trades::test_set_fee_share_no_role_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_transfers::test_deposit (l1_gas: ~0, l1_data_gas: ~2016, l2_gas: ~6810880)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_trades::test_settle_trade_configurator_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4379200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_set_oracle_contract_no_role_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::perpetual::tests::test_perp_option::test_update_perpetual_option_margin_params_cannot_update_non_existing_params (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_trades::test_settle_market_mismatch (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4379200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_set_oracle_contract_executor_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear::test_set_oracle_contract_state_publisher_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_trades::test_settle_trade_no_role_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4379200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_liquidations::test_get_insurance_fund_account (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4179200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_liquidations::test_account_liquidate_liquidator_cannot_be_liquidated (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_liquidations::test_get_liquidate_fee (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4139200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_liquidations::test_account_liquidate_no_role_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_liquidations::test_set_insurance_fund_account (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4979200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_liquidations::test_set_insurance_fund_account_configurator_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_transfers::test_deposit_on_behalf_of (l1_gas: ~0, l1_data_gas: ~2016, l2_gas: ~6810880)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_liquidations::test_account_liquidate_invalid_share (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_trades::test_set_fee_share_executor_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_trades::test_set_fee_share (l1_gas: ~0, l1_data_gas: ~1344, l2_gas: ~5369920)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_liquidations::test_account_liquidate_state_publisher_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_liquidations::test_set_insurance_fund_account_executor_unauthorized (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~4339200)
[PASS] paradex_paraclear::paraclear::tests::test_paraclear_trades::test_settle_trade (l1_gas: ~0, l1_data_gas: ~3648, l2_gas: ~26799040)
Tests: 202 passed, 0 failed, 0 skipped, 2 ignored, 0 filtered out


Collected 44 test(s) from paradex_registry package
Running 44 test(s) from tests/
[IGNORE] paradex_registry::tests::test_registry::testRegistry::test_message_for_transfer_restriction
[PASS] paradex_registry::tests::test_registry::testRegistry::get_name (l1_gas: ~0, l1_data_gas: ~576, l2_gas: ~705600)
[PASS] paradex_registry::tests::test_registry::testRegistry::get_version (l1_gas: ~0, l1_data_gas: ~576, l2_gas: ~705600)
[PASS] paradex_registry::tests::test_registry::testRegistry::test_detect_account_transfer_restriction_from_accounts (l1_gas: ~0, l1_data_gas: ~5856, l2_gas: ~12807680)
```

```
[PASS] paradex_registry::tests::test_registry::testRegistry::
    test_detect_account_transfer_restriction_with_withdrawals_paused (l1_gas: ~0, l1_data_gas: ~5952, l2_gas:
    ~14607680)
[PASS] paradex_registry::tests::test_registry::testRegistry::
    test_detect_account_transfer_restriction_with_withdrawals_unpaused (l1_gas: ~0, l1_data_gas: ~5856, l2_gas:
    ~15167680)
[PASS] paradex_registry::tests::test_registry::testRegistry::test_detect_transfer_restriction (l1_gas: ~0,
    l1_data_gas: ~1632, l2_gas: ~18952640)
[PASS] paradex_registry::tests::test_registry::testRegistry::test_detect_transfer_restriction_from_auxiliary (
    l1_gas: ~0, l1_data_gas: ~1632, l2_gas: ~17832640)
[PASS] paradex_registry::tests::test_registry::testRegistry::
    test_detect_account_transfer_restriction_with_authorization (l1_gas: ~0, l1_data_gas: ~5856, l2_gas:
    ~14567680)
[PASS] paradex_registry::tests::test_registry::testRegistry::
    test_detect_account_transfer_restriction_without_authorization (l1_gas: ~0, l1_data_gas: ~5856, l2_gas:
    ~18967680)
[PASS] paradex_registry::tests::test_registry::testRegistry::test_detect_transfer_restriction_from_normal_account
     (l1_gas: ~0, l1_data_gas: ~1632, l2_gas: ~17832640)
[PASS] paradex_registry::tests::test_registry::testRegistry::test_operator_cannot_directly_transfer_to_auxiliary
    (l1_gas: ~0, l1_data_gas: ~960, l2_gas: ~1866560)
[PASS] paradex_registry::tests::test_registry::testRegistry::test_detect_transfer_restriction_from_operator (
    l1_gas: ~0, l1_data_gas: ~1632, l2_gas: ~17832640)
[PASS] paradex_registry::tests::test_registry::testRegistry::test_operator_cannot_directly_transfer_to_vault (
    l1_gas: ~0, l1_data_gas: ~768, l2_gas: ~1606080)
[PASS] paradex_registry::tests::test_registry::testRegistry::test_detect_transfer_restriction_from_owner (l1_gas:
     ~0, l1_data_gas: ~1632, l2_gas: ~17832640)
[PASS] paradex_registry::tests::test_registry::testRegistry::
    test_other_vaults_cannot_trigger_transfer_from_operator_to_auxiliary (l1_gas: ~0, l1_data_gas: ~960, l2_gas:
    ~1866560)
[PASS] paradex_registry::tests::test_registry::testRegistry::
    test_other_vaults_cannot_trigger_transfer_from_operator_to_vault (l1_gas: ~0, l1_data_gas: ~768, l2_gas:
    ~1606080)
[PASS] paradex_registry::tests::test_registry::testRegistry::test_detect_transfer_restriction_from_vault (l1_gas:
     ~0, l1_data_gas: ~1632, l2_gas: ~19232640)
[PASS] paradex_registry::tests::test_registry::testRegistry::test_register_auxiliary_account (l1_gas: ~0,
    l1_data_gas: ~768, l2_gas: ~1446080)
[PASS] paradex_registry::tests::test_registry::testRegistry::test_register_sub_operator (l1_gas: ~0, l1_data_gas:
     ~1440, l2_gas: ~4012160)
[PASS] paradex_registry::tests::test_registry::testRegistry::test_register_sub_operator_expired_signature (l1_gas
    : ~0, l1_data_gas: ~1056, l2_gas: ~2306560)
[PASS] paradex_registry::tests::test_registry::testRegistry::test_pause_all_vault_withdrawals (l1_gas: ~0,
    l1_data_gas: ~672, l2_gas: ~1705600)
[PASS] paradex_registry::tests::test_registry::testRegistry::test_register_sub_operator_already_registered (
    l1_gas: ~0, l1_data_gas: ~1440, l2_gas: ~4997760)
[PASS] paradex_registry::tests::test_registry::testRegistry::test_register_sub_operator_invalid_signature (l1_gas
    : ~0, l1_data_gas: ~1152, l2_gas: ~2426560)
[PASS] paradex_registry::tests::test_registry::testRegistry::test_register_sub_operator_max_limit (l1_gas: ~0,
    l1_data_gas: ~1056, l2_gas: ~3066560)
[PASS] paradex_registry::tests::test_registry::testRegistry::test_register_sub_operator_unauthorized (l1_gas: ~0,
     l1_data_gas: ~1056, l2_gas: ~1746560)
[PASS] paradex_registry::tests::test_registry::testRegistry::test_register_vault (l1_gas: ~0, l1_data_gas: ~768,
    l2_gas: ~1446080)
[PASS] paradex_registry::tests::test_registry::testRegistry::test_register_vault_unauthorized (l1_gas: ~0,
    l1_data_gas: ~576, l2_gas: ~705600)
[PASS] paradex_registry::tests::test_registry::testRegistry::test_unpause_all_vault_withdrawals (l1_gas: ~0,
    l1_data_gas: ~576, l2_gas: ~1705600)
[PASS] paradex_registry::tests::test_registry::testRegistry::test_registry_initialization (l1_gas: ~0,
    l1_data_gas: ~576, l2_gas: ~865600)
[PASS] paradex_registry::tests::test_registry::testRegistry::test_register_sub_operator_same_nonce (l1_gas: ~0,
    l1_data_gas: ~1440, l2_gas: ~3372160)
[PASS] paradex_registry::tests::test_registry::testRegistry::
    test_unregister_sub_operator_suboperator_not_registered (l1_gas: ~0, l1_data_gas: ~1056, l2_gas: ~1946560)
[PASS] paradex_registry::tests::test_registry::testRegistry::
    test_vault_can_trigger_transfer_from_operator_to_auxiliary (l1_gas: ~0, l1_data_gas: ~960, l2_gas: ~2666560)
[PASS] paradex_registry::tests::test_registry::testRegistry::test_unregister_one_sub_operator_last_sub_operator (
    l1_gas: ~0, l1_data_gas: ~2112, l2_gas: ~8278720)
[PASS] paradex_registry::tests::test_registry::testRegistry::test_unregister_sub_operator (l1_gas: ~0,
    l1_data_gas: ~1440, l2_gas: ~5512640)
[PASS] paradex_registry::tests::test_registry::testRegistry::
    test_vault_can_trigger_transfer_from_operator_to_vault (l1_gas: ~0, l1_data_gas: ~768, l2_gas: ~2406080)
```

```
[PASS] paradex_registry::tests::test_registry::testRegistry::test_unregister_sub_operator_unauthorized (l1_gas: ~0, l1_data_gas: ~1056, l2_gas: ~1506560)
[PASS] paradex_registry::tests::test_registry::testRegistry::test_unregister_sub_operator_not_registered_for_vault (l1_gas: ~0, l1_data_gas: ~1632, l2_gas: ~3672640)
[PASS] paradex_registry::tests::test_registry::testRegistry::test_unregister_one_sub_operator_of_multiple (l1_gas: ~0, l1_data_gas: ~2112, l2_gas: ~8278720)
[PASS] paradex_registry::warden::tests::test_warden::test_increment_trades_unauthorized (l1_gas: ~0, l1_data_gas: ~768, l2_gas: ~2506560)
[PASS] paradex_registry::warden::tests::test_warden::test_initialization (l1_gas: ~0, l1_data_gas: ~672, l2_gas: ~1640960)
[PASS] paradex_registry::warden::tests::test_warden::test_initialization_unauthorized (l1_gas: ~0, l1_data_gas: ~576, l2_gas: ~905600)
[PASS] paradex_registry::warden::tests::test_warden::test_can_transfer (l1_gas: ~0, l1_data_gas: ~1056, l2_gas: ~8996480)
[PASS] paradex_registry::warden::tests::test_warden::test_trade_tracking (l1_gas: ~0, l1_data_gas: ~1056, l2_gas: ~6316480)
Tests: 43 passed, 0 failed, 0 skipped, 1 ignored, 0 filtered out


Collected 0 test(s) from paradex_test_common package
Running 0 test(s) from tests/
Tests: 0 passed, 0 failed, 0 skipped, 0 ignored, 0 filtered out


Collected 121 test(s) from paradex_vaults package
Running 121 test(s) from tests/
[IGNORE] paradex_vaults::tests::test_vault::testVault::test_internal_convert_to_assets
[IGNORE] paradex_vaults::tests::test_vault_deposit::testVaultDeposit::test_deposits_with_increasing_account_value
[IGNORE] paradex_vaults::tests::test_vault::testVault::test_internal_convert_to_shares
[PASS] paradex_vaults::tests::test_factory::testFactory::deploy (l1_gas: ~0, l1_data_gas: ~2784, l2_gas: ~3492160)
[PASS] paradex_vaults::tests::test_factory::testFactory::deploy_vault_profit_share_above_max_profit_share (l1_gas: ~0, l1_data_gas: ~2784, l2_gas: ~3987520)
[PASS] paradex_vaults::tests::test_factory::testFactory::deploy_vault_profit_share_percentage_below_max_profit_share (l1_gas: ~0, l1_data_gas: ~4512, l2_gas: ~10030400)
[PASS] paradex_vaults::tests::test_factory::testFactory::deploy_vault_lockup_period_above_max_lockup_period (l1_gas: ~0, l1_data_gas: ~2784, l2_gas: ~3947520)
[PASS] paradex_vaults::tests::test_factory::testFactory::deploy_vault_lockup_below_max_lockup_period (l1_gas: ~0, l1_data_gas: ~4512, l2_gas: ~9990400)
[PASS] paradex_vaults::tests::test_factory::testFactory::deploy_vault_profit_share_percentage_equal_than_max_profit_share (l1_gas: ~0, l1_data_gas: ~4512, l2_gas: ~10030400)
[PASS] paradex_vaults::tests::test_factory::testFactory::deploy_vault_lockup_equal_than_max_lockup_period (l1_gas: ~0, l1_data_gas: ~4512, l2_gas: ~9990400)
[PASS] paradex_vaults::tests::test_factory::testFactory::deploy_vault_lockup_equal_than_zero (l1_gas: ~0, l1_data_gas: ~4416, l2_gas: ~9990400)
[PASS] paradex_vaults::tests::test_factory::testFactory::deploy_vault_profit_share_percentage_equal_than_zero (l1_gas: ~0, l1_data_gas: ~4416, l2_gas: ~10030400)
[PASS] paradex_vaults::tests::test_factory::testFactory::deploy_vault_test (l1_gas: ~0, l1_data_gas: ~4704, l2_gas: ~10575040)
[PASS] paradex_vaults::tests::test_factory::testFactory::deploy_vault_tvl_limit_equal_than_min_initial_deposit (l1_gas: ~0, l1_data_gas: ~4512, l2_gas: ~10030400)
[PASS] paradex_vaults::tests::test_factory::testFactory::deploy_vault_tvl_limit_above_min_initial_deposit (l1_gas: ~0, l1_data_gas: ~4512, l2_gas: ~10030400)
[PASS] paradex_vaults::tests::test_factory::testFactory::deploy_vault_tvl_limit_below_min_initial_deposit (l1_gas: ~0, l1_data_gas: ~2784, l2_gas: ~3947520)
[PASS] paradex_vaults::tests::test_factory::testFactory::deploy_vault_tvl_limit_equal_than_zero (l1_gas: ~0, l1_data_gas: ~4416, l2_gas: ~9990400)
[PASS] paradex_vaults::tests::test_factory::testFactory::get_name (l1_gas: ~0, l1_data_gas: ~2784, l2_gas: ~3452160)
[PASS] paradex_vaults::tests::test_factory::testFactory::deploy_vault_wrong_caller (l1_gas: ~0, l1_data_gas: ~2784, l2_gas: ~3452160)
[PASS] paradex_vaults::tests::test_factory::testFactory::get_version (l1_gas: ~0, l1_data_gas: ~2784, l2_gas: ~3452160)
[PASS] paradex_vaults::tests::test_factory::testFactory::set_min_owner_share_percentage_too_high (l1_gas: ~0, l1_data_gas: ~2784, l2_gas: ~3652160)
[PASS] paradex_vaults::tests::test_factory::testFactory::set_max_lockup_period_seconds (l1_gas: ~0, l1_data_gas: ~2784, l2_gas: ~4627520)
```

```
[PASS] paradex_vaults::tests::test_factory::testFactory::set_vault_class_hash (l1_gas: ~0, l1_data_gas: ~2784, l2_gas: ~4627520)
[PASS] paradex_vaults::tests::test_factory::testFactory::set_max_lockup_period_wrong_caller (l1_gas: ~0, l1_data_gas: ~2784, l2_gas: ~3452160)
[PASS] paradex_vaults::tests::test_factory::testFactory::set_max_lockup_period_zero_hash (l1_gas: ~0, l1_data_gas: ~2784, l2_gas: ~3652160)
[PASS] paradex_vaults::tests::test_factory::testFactory::set_vault_class_hash_wrong_caller (l1_gas: ~0, l1_data_gas: ~2784, l2_gas: ~3452160)
[PASS] paradex_vaults::tests::test_factory::testFactory::set_vault_class_hash_zero_hash (l1_gas: ~0, l1_data_gas: ~2784, l2_gas: ~3692160)
[PASS] paradex_vaults::tests::test_factory::testFactory::set_max_profit_share_percentage (l1_gas: ~0, l1_data_gas: ~2784, l2_gas: ~4627520)
[PASS] paradex_vaults::tests::test_factory::testFactory::test_set_and_get_registry (l1_gas: ~0, l1_data_gas: ~2784, l2_gas: ~4132160)
[PASS] paradex_vaults::tests::test_factory::testFactory::set_max_profit_share_percentage_too_high (l1_gas: ~0, l1_data_gas: ~2784, l2_gas: ~3652160)
[PASS] paradex_vaults::tests::test_vault::testVault::set_tvl_limit (l1_gas: ~0, l1_data_gas: ~4704, l2_gas: ~11115520)
[PASS] paradex_vaults::tests::test_factory::testFactory::set_max_profit_share_percentage_wrong_caller (l1_gas: ~0, l1_data_gas: ~2784, l2_gas: ~3452160)
[PASS] paradex_vaults::tests::test_vault::testVault::test_authorized_transfer (l1_gas: ~0, l1_data_gas: ~0, l2_gas: ~320000)
[PASS] paradex_vaults::tests::test_factory::testFactory::set_max_profit_share_percentage_zero (l1_gas: ~0, l1_data_gas: ~2784, l2_gas: ~3652160)
[PASS] paradex_vaults::tests::test_vault::testVault::set_tvl_limit_wrong_caller (l1_gas: ~0, l1_data_gas: ~4704, l2_gas: ~9935040)
[PASS] paradex_vaults::tests::test_vault::testVault::set_tvl_limit_zero (l1_gas: ~0, l1_data_gas: ~4608, l2_gas: ~11115520)
[PASS] paradex_vaults::tests::test_factory::testFactory::set_min_initial_deposit (l1_gas: ~0, l1_data_gas: ~2784, l2_gas: ~4627520)
[PASS] paradex_vaults::tests::test_factory::testFactory::set_min_initial_deposit_wrong_caller (l1_gas: ~0, l1_data_gas: ~2784, l2_gas: ~3452160)
[PASS] paradex_vaults::tests::test_factory::testFactory::set_min_initial_deposit_zero_hash (l1_gas: ~0, l1_data_gas: ~2784, l2_gas: ~3652160)
[PASS] paradex_vaults::tests::test_vault::testVault::test_avg_deposit_time_not_deposited (l1_gas: ~0, l1_data_gas: ~4704, l2_gas: ~10015040)
[PASS] paradex_vaults::tests::test_vault::testVault::test_avg_deposit_time_deposited (l1_gas: ~0, l1_data_gas: ~5376, l2_gas: ~19785920)
[PASS] paradex_vaults::tests::test_factory::testFactory::set_min_owner_share__percentagewrong_caller (l1_gas: ~0, l1_data_gas: ~2784, l2_gas: ~3452160)
[PASS] paradex_vaults::tests::test_factory::testFactory::set_min_owner_share_percentage (l1_gas: ~0, l1_data_gas: ~2784, l2_gas: ~4627520)
[PASS] paradex_vaults::tests::test_vault::testVault::test_burn (l1_gas: ~0, l1_data_gas: ~5280, l2_gas: ~25106880)
[PASS] paradex_vaults::tests::test_factory::testFactory::set_min_owner_share_percentage_can_be_zero (l1_gas: ~0, l1_data_gas: ~2688, l2_gas: ~4627520)
[PASS] paradex_vaults::tests::test_factory::testFactory::upgrade_factory (l1_gas: ~0, l1_data_gas: ~2784, l2_gas: ~4147520)
[PASS] paradex_vaults::tests::test_vault::testVault::test_burn_admin (l1_gas: ~0, l1_data_gas: ~5280, l2_gas: ~20826880)
[PASS] paradex_vaults::tests::test_vault::testVault::test_burn_admin_zero_amount (l1_gas: ~0, l1_data_gas: ~4704, l2_gas: ~10175040)
[PASS] paradex_vaults::tests::test_vault::testVault::test_set_withdrawal_mode_standard_again (l1_gas: ~0, l1_data_gas: ~5376, l2_gas: ~13381120)
[PASS] paradex_vaults::tests::test_vault::testVault::test_burn_more_than_balance (l1_gas: ~0, l1_data_gas: ~5280, l2_gas: ~19465920)
[PASS] paradex_vaults::tests::test_vault::testVault::test_burn_admin_unauthorized (l1_gas: ~0, l1_data_gas: ~5280, l2_gas: ~19505920)
[PASS] paradex_vaults::tests::test_factory::testFactory::upgrade_factory_wrong_caller (l1_gas: ~0, l1_data_gas: ~2784, l2_gas: ~3492160)
[PASS] paradex_vaults::tests::test_vault::testVault::test_set_withdrawal_mode_unauthorized (l1_gas: ~0, l1_data_gas: ~4704, l2_gas: ~10295040)
[PASS] paradex_vaults::tests::test_factory::testFactory::upgrade_factory_zero_class_hash (l1_gas: ~0, l1_data_gas: ~2784, l2_gas: ~3692160)
[PASS] paradex_vaults::tests::test_vault::testVault::get_decimals (l1_gas: ~0, l1_data_gas: ~4704, l2_gas: ~9975040)
[PASS] paradex_vaults::tests::test_vault::testVault::get_version (l1_gas: ~0, l1_data_gas: ~4704, l2_gas: ~9975040)
[PASS] paradex_vaults::tests::test_vault::testVault::is_initialised (l1_gas: ~0, l1_data_gas: ~4704, l2_gas: ~10015040)
```

```
[PASS] paradex_vaults::tests::test_vault::testVault::set_lockup_period_limit_wrong_caller (l1_gas: ~0, l1_data_gas: ~4704, l2_gas: ~9935040)
[PASS] paradex_vaults::tests::test_vault::testVault::set_lockup_period_reduce_value (l1_gas: ~0, l1_data_gas: ~4704, l2_gas: ~11155520)
[PASS] paradex_vaults::tests::test_vault::testVault::set_lockup_period_limit_cannot_increase (l1_gas: ~0, l1_data_gas: ~4704, l2_gas: ~10415040)
[PASS] paradex_vaults::tests::test_vault::testVault::set_lockup_period_same_value (l1_gas: ~0, l1_data_gas: ~4704, l2_gas: ~11155520)
[PASS] paradex_vaults::tests::test_vault::testVault::upgrade_vault (l1_gas: ~0, l1_data_gas: ~4704, l2_gas: ~10670400)
[PASS] paradex_vaults::tests::test_vault::testVault::set_lockup_period_zero_value (l1_gas: ~0, l1_data_gas: ~4608, l2_gas: ~11155520)
[PASS] paradex_vaults::tests::test_vault::testVault::test_total_assets_with_multi_strategy (l1_gas: ~0, l1_data_gas: ~6144, l2_gas: ~28832960)
[PASS] paradex_vaults::tests::test_vault::testVault::test_set_withdrawal_mode_fast_non_multi_strategy (l1_gas: ~0, l1_data_gas: ~4704, l2_gas: ~10375040)
[PASS] paradex_vaults::tests::test_vault::testVault::upgrade_vault_zero_class_hash (l1_gas: ~0, l1_data_gas: ~4704, l2_gas: ~10175040)
[PASS] paradex_vaults::tests::test_vault::testVault::upgrade_vault_wrong_caller (l1_gas: ~0, l1_data_gas: ~4704, l2_gas: ~9975040)
[PASS] paradex_vaults::tests::test_vault_close::testVaultClose::test_close_vault_other_users_cannot_close (l1_gas: ~0, l1_data_gas: ~4704, l2_gas: ~10175040)
[PASS] paradex_vaults::tests::test_vault_close::testVaultClose::test_close_vault_with_open_positions (l1_gas: ~0, l1_data_gas: ~5760, l2_gas: ~11655040)
[PASS] paradex_vaults::tests::test_vault_deposit::testVaultDeposit::test_deposit_when_withdrawals_restricted (l1_gas: ~0, l1_data_gas: ~5376, l2_gas: ~20105920)
[PASS] paradex_vaults::tests::test_vault_close::testVaultClose::test_close_vault_with_open_positions_sub_operator (l1_gas: ~0, l1_data_gas: ~6240, l2_gas: ~13381120)
[PASS] paradex_vaults::tests::test_vault_close::testVaultClose::test_close_vault_wrong_caller (l1_gas: ~0, l1_data_gas: ~4704, l2_gas: ~9935040)
[PASS] paradex_vaults::tests::test_vault_deposit::testVaultDeposit::test_donate (l1_gas: ~0, l1_data_gas: ~4992, l2_gas: ~20985920)
[PASS] paradex_vaults::tests::test_vault_close::testVaultClose::test_close_vault_with_sub_operators (l1_gas: ~0, l1_data_gas: ~6432, l2_gas: ~33248320)
[PASS] paradex_vaults::tests::test_vault_close::testVaultClose::test_close_when_withdrawals_restricted (l1_gas: ~0, l1_data_gas: ~5376, l2_gas: ~23561280)
[PASS] paradex_vaults::tests::test_vault_deposit::testVaultDeposit::test_multiple_deposits_from_different_accounts (l1_gas: ~0, l1_data_gas: ~5856, l2_gas: ~53941440)
[PASS] paradex_vaults::tests::test_vault_deposit::testVaultDeposit::test_other_users_cannot_deposit_before_owner (l1_gas: ~0, l1_data_gas: ~4800, l2_gas: ~11416000)
[PASS] paradex_vaults::tests::test_vault_deposit::testVaultDeposit::test_multiple_deposits_per_account (l1_gas: ~0, l1_data_gas: ~5664, l2_gas: ~64353280)
[PASS] paradex_vaults::tests::test_vault_withdraw::testVaultWithdraw::test_request_withdrawal_before_lockup (l1_gas: ~0, l1_data_gas: ~5280, l2_gas: ~20265920)
[PASS] paradex_vaults::tests::test_vault_deposit::testVaultDeposit::test_owner_can_deposit_less_than_min_initial_deposit_after_first_deposit (l1_gas: ~0, l1_data_gas: ~5280, l2_gas: ~31236800)
[PASS] paradex_vaults::tests::test_vault_deposit::testVaultDeposit::test_owner_cannot_deposit_less_than_min_initial_deposit_at_first_deposit (l1_gas: ~0, l1_data_gas: ~4800, l2_gas: ~11456000)
[PASS] paradex_vaults::tests::test_vault_close::testVaultClose::test_close_vault (runs: 22, (l1_gas: {max: ~0, min: ~0, mean: ~0.00, std deviation: ~0.00}, l1_data_gas: {max: ~5472, min: ~5472, mean: ~5472.00, std deviation: ~0.00}, l2_gas: {max: ~32481280, min: ~32401280, mean: ~32450370.91, std deviation: ~20650.58}))
[PASS] paradex_vaults::tests::test_vault_close::testVaultClose::test_close_vault_already_closed (l1_gas: ~0, l1_data_gas: ~4704, l2_gas: ~12110400)
[PASS] paradex_vaults::tests::test_vault_close::testVaultClose::test_close_vault_operator_cannot_close (l1_gas: ~0, l1_data_gas: ~4704, l2_gas: ~10295040)
request_withdrawal: 500000000
lp_shares_owner: 50000000
lp_shares_receiver: 500000000
[PASS] paradex_vaults::tests::test_vault_withdraw::testVaultWithdraw::test_request_withdrawal_before_lockup_period_allowed_if_closed (l1_gas: ~0, l1_data_gas: ~5664, l2_gas: ~34629760)
[PASS] paradex_vaults::tests::test_vault::testVault::test_internal_total_assets (runs: 22, (l1_gas: {max: ~0, min: ~0, mean: ~0.00, std deviation: ~0.00}, l1_data_gas: {max: ~5280, min: ~5280, mean: ~5280.00, std deviation: ~0.00}, l2_gas: {max: ~28721280, min: ~28681280, mean: ~28686734.55, std deviation: ~13726.97}))
vault_testing_state initialized
initial_deposit: 1000000000
second_deposit: 2000000000
withdraw: 3000000000
```

```
vault_testing_state initialized
deposit/withdraw: @DepositWithdrawTestStruct { step: 1, kind: 1, amount: 50000000, timestamp: 1727428432, expected_asset_weighted_time: 1727428432, expected_asset_balance: 50000000 }
[PASS] paradex_vaults::tests::test_vault_deposit::testVaultDeposit::test_average_holding_time (l1_gas: ~0, l1_data_gas: ~4864, l2_gas: ~38821760)
deposit/withdraw: @DepositWithdrawTestStruct { step: 2, kind: 1, amount: 20000000, timestamp: 1727429836, expected_asset_weighted_time: 1727428833, expected_asset_balance: 70000000 }
deposit/withdraw: @DepositWithdrawTestStruct { step: 3, kind: 1, amount: 35000000, timestamp: 1727506543, expected_asset_weighted_time: 1727454736, expected_asset_balance: 105000000 }
deposit/withdraw: @DepositWithdrawTestStruct { step: 4, kind: 1, amount: 15000000, timestamp: 1727534910, expected_asset_weighted_time: 1727464757, expected_asset_balance: 120000000 }
deposit/withdraw: @DepositWithdrawTestStruct { step: 5, kind: 2, amount: 20000000, timestamp: 1727579447, expected_asset_weighted_time: 1727483872, expected_asset_balance: 100000000 }
deposit/withdraw: @DepositWithdrawTestStruct { step: 6, kind: 1, amount: 50000000, timestamp: 1727579490, expected_asset_weighted_time: 1727515744, expected_asset_balance: 150000000 }
deposit/withdraw: @DepositWithdrawTestStruct { step: 7, kind: 1, amount: 25000000, timestamp: 1727584002, expected_asset_weighted_time: 1727525495, expected_asset_balance: 175000000 }
deposit/withdraw: @DepositWithdrawTestStruct { step: 8, kind: 2, amount: 175000000, timestamp: 1727614897, expected_asset_weighted_time: 0, expected_asset_balance: 0 }
deposit/withdraw: @DepositWithdrawTestStruct { step: 9, kind: 1, amount: 20000000, timestamp: 1727614897, expected_asset_weighted_time: 1727614897, expected_asset_balance: 20000000 }
[PASS] paradex_vaults::tests::test_vault_withdraw::testVaultWithdraw::test_request_withdrawal_min_owner_share_no_depositors (l1_gas: ~0, l1_data_gas: ~5664, l2_gas: ~28893440)
[PASS] paradex_vaults::tests::test_vault_deposit::testVaultDeposit::test_average_holding_time_two_withdrawals (l1_gas: ~0, l1_data_gas: ~4864, l2_gas: ~79403840)
[PASS] paradex_vaults::tests::test_vault_withdraw::testVaultWithdraw::test_request_withdrawal (runs: 22, (l1_gas: {max: ~0, min: ~0, mean: ~0.00, std deviation: ~0.00}, l1_data_gas: {max: ~5664, min: ~5664, mean: ~5664.00, std deviation: ~0.00}, l2_gas: {max: ~35293440, min: ~35213440, mean: ~35262530.91, std deviation: ~20650.58}))
[PASS] paradex_vaults::tests::test_vault_withdraw::testVaultWithdraw::test_request_withdrawal_min_owner_share_no_depositors_below_min_initial_deposit (l1_gas: ~0, l1_data_gas: ~5280, l2_gas: ~20265920)
request_withdrawal: 50000000
[PASS] paradex_vaults::tests::test_vault_withdraw::testVaultWithdraw::test_request_withdrawal_restricted (l1_gas: ~0, l1_data_gas: ~5376, l2_gas: ~20385920)
lp_shares_receiver: 50000000
[PASS] paradex_vaults::tests::test_vault_withdraw::testVaultWithdraw::test_request_withdrawal_mode_fast (l1_gas: ~0, l1_data_gas: ~7104, l2_gas: ~43834880)
request_withdrawal: 95000000
lp_shares_receiver: 5000000
[PASS] paradex_vaults::tests::test_vault_withdraw::testVaultWithdraw::test_request_withdrawal_over_shares_balance (l1_gas: ~0, l1_data_gas: ~5280, l2_gas: ~19745920)
[PASS] paradex_vaults::tests::test_vault_withdraw::testVaultWithdraw::test_request_withdrawal_mode_fast_not_enough_balance_on_main_strategy (l1_gas: ~0, l1_data_gas: ~7104, l2_gas: ~44514880)
[PASS] paradex_vaults::tests::test_vault_withdraw::testVaultWithdraw::test_request_withdrawal_restricted_operator (l1_gas: ~0, l1_data_gas: ~5376, l2_gas: ~20665920)
[PASS] paradex_vaults::tests::test_vault_withdraw::testVaultWithdraw::test_request_withdrawal_closed_vault (runs: 22, (l1_gas: {max: ~0, min: ~0, mean: ~0.00, std deviation: ~0.00}, l1_data_gas: {max: ~5664, min: ~5664, mean: ~5664.00, std deviation: ~0.00}, l2_gas: {max: ~42268800, min: ~42188800, mean: ~42237890.91, std deviation: ~20650.58}))
[PASS] paradex_vaults::tests::test_vault_withdraw::testVaultWithdraw::test_request_withdrawal_restricted_owner (l1_gas: ~0, l1_data_gas: ~5376, l2_gas: ~20385920)
[PASS] paradex_vaults::tests::test_vault_withdraw::testVaultWithdraw::test_request_withdrawal_min_owner_share_below_min_initial_deposit (l1_gas: ~0, l1_data_gas: ~5472, l2_gas: ~29996800)
[PASS] paradex_vaults::tests::test_vault_withdraw::testVaultWithdraw::test_request_withdrawal_min_owner_share_exceeded (l1_gas: ~0, l1_data_gas: ~5472, l2_gas: ~29956800)
[PASS] paradex_vaults::tests::test_vault_withdraw::testVaultWithdraw::test_request_withdrawal_unhealthy_vault (l1_gas: ~0, l1_data_gas: ~5568, l2_gas: ~21166400)
[PASS] paradex_vaults::tests::test_vault_withdraw::testVaultWithdraw::test_request_withdrawal_restricted_sub_operator (l1_gas: ~0, l1_data_gas: ~6048, l2_gas: ~24192000)
[PASS] paradex_vaults::tests::test_vault_deposit::testVaultDeposit::test_deposit (runs: 22, (l1_gas: {max: ~0, min: ~0, mean: ~0.00, std deviation: ~0.00}, l1_data_gas: {max: ~5280, min: ~5280, mean: ~5280.00, std deviation: ~0.00}, l2_gas: {max: ~22545920, min: ~22545920, mean: ~22545920.00, std deviation: ~0.00}))
[PASS] paradex_vaults::tests::test_vault_withdraw::testVaultWithdraw::test_request_withdrawal_unrestricted (l1_gas: ~0, l1_data_gas: ~5664, l2_gas: ~27053440)
request_withdrawal: 500000
lp_shares_owner: 50000
lp_shares_receiver: 500000
```

```
[PASS] paradex_vaults::tests::test_vault_withdraw::testVaultWithdraw::
    test_request_withdrawal_with_profit_share_paid_to_owner (l1_gas: ~0, l1_data_gas: ~5760, l2_gas: ~34149760)
[PASS] paradex_vaults::tests::test_vault_withdraw::testVaultWithdraw::test_token_transfer (l1_gas: ~0,
    l1_data_gas: ~5376, l2_gas: ~21506880)
[PASS] paradex_vaults::tests::test_vault_withdraw::testVaultWithdraw::test_token_transfer_before_lockup (l1_gas:
    ~0, l1_data_gas: ~5280, l2_gas: ~20065920)
[PASS] paradex_vaults::tests::test_vault_withdraw::testVaultWithdraw::test_token_transfer_insufficient_balance (
    l1_gas: ~0, l1_data_gas: ~5280, l2_gas: ~20065920)
[PASS] paradex_vaults::tests::test_vault_withdraw::testVaultWithdraw::
    test_token_transfer_min_owner_share_below_min_initial_deposit (l1_gas: ~0, l1_data_gas: ~5280, l2_gas:
    ~20705920)
[PASS] paradex_vaults::tests::test_vault_deposit::testVaultDeposit::test_deposit_liquidated_vault (l1_gas: ~0,
    l1_data_gas: ~5376, l2_gas: ~22307840)
[PASS] paradex_vaults::tests::test_vault_withdraw::testVaultWithdraw::
    test_token_transfer_min_owner_share_exceeded (l1_gas: ~0, l1_data_gas: ~5472, l2_gas: ~30076800)
[PASS] paradex_vaults::tests::test_vault::testVault::test_internal_total_assets_with_multi_strategy (runs: 20, (
    l1_gas: {max: ~0, min: ~0, mean: ~0.00, std deviation: ~0.00}, l1_data_gas: {max: ~5952, min: ~5952, mean:
    ~5952.00, std deviation: ~0.00}, l2_gas: {max: ~37847360, min: ~37807360, mean: ~37819360.00, std deviation:
    ~18330.30}))
[PASS] paradex_vaults::tests::test_vault_withdraw::testVaultWithdraw::test_token_transfer_restricted (l1_gas: ~0,
     l1_data_gas: ~5376, l2_gas: ~20505920)
request_withdrawal: 5000000000000000000
getAccountValue: 308528156332618400553265
[PASS] paradex_vaults::tests::test_vault_withdraw::testVaultWithdraw::test_withdraw_to_auxiliary_account (l1_gas:
     ~0, l1_data_gas: ~5664, l2_gas: ~28573440)
shares_to_burn: 101000000
[PASS] paradex_vaults::tests::test_vault::testVault::test_owner_burn_below_min_initial_deposit (l1_gas: ~0,
    l1_data_gas: ~5472, l2_gas: ~30276800)
shares_to_burn: 100000000
[PASS] paradex_vaults::tests::test_vault::testVault::test_owner_burn_below_min_owner_share (l1_gas: ~0,
    l1_data_gas: ~5472, l2_gas: ~30276800)
[PASS] paradex_vaults::tests::test_vault_withdraw::testVaultWithdraw::test_request_withdrawal_zero_shares (l1_gas
    : ~0, l1_data_gas: ~5280, l2_gas: ~19745920)
[PASS] paradex_vaults::tests::test_vault::testVault::test_set_withdrawal_mode_fast (l1_gas: ~0, l1_data_gas:
    ~5472, l2_gas: ~13181120)
[PASS] paradex_vaults::tests::test_vault_deposit::testVaultDeposit::test_deposit_unhealthy_vault (l1_gas: ~0,
    l1_data_gas: ~4896, l2_gas: ~11896000)
[PASS] paradex_vaults::tests::test_vault_withdraw::testVaultWithdraw::test_request_withdrawal_with_sub_operators
    (runs: 22, (l1_gas: {max: ~0, min: ~0, mean: ~0.00, std deviation: ~0.00}, l1_data_gas: {max: ~6432, min:
    ~6432, mean: ~6432.00, std deviation: ~0.00}, l2_gas: {max: ~42940480, min: ~42860480, mean: ~42909570.91,
    std deviation: ~20650.58}))
[PASS] paradex_vaults::tests::test_vault_deposit::testVaultDeposit::test_deposit_tvl_limit_reached (runs: 22, (
    l1_gas: {max: ~0, min: ~0, mean: ~0.00, std deviation: ~0.00}, l1_data_gas: {max: ~4704, min: ~4704, mean:
    ~4704.00, std deviation: ~0.00}, l2_gas: {max: ~13575040, min: ~13535040, mean: ~13540494.55, std deviation:
    ~13726.97}))
Tests: 118 passed, 0 failed, 0 skipped, 3 ignored, 0 filtered out
```