Blockchain Protocol Security Analysis Report

Customer: Polymesh

Date: 13/11/2024

We express our gratitude to the Polymesh team for the collaborative engagement that enabled the execution of this Blockchain Protocol Security Assessment.

Polymesh is a Layer 1 blockchain specifically designed for security tokens, emphasizing compliance within regulated markets. It facilitates the issuance, transfer, and management of security tokens while ensuring adherence to regulatory standards.

Key features include identity verification mechanisms, governance frameworks, and compliance tools that enhance liquidity and create new funding opportunities for organizations. By prioritizing regulatory compliance and security, Polymesh aims to revolutionize the landscape of asset tokenization.

Document

Name	Blockchain Protocol Review and Security Analysis Report for Polymesh
Audited By	Nino Lipartiia, Tanuj Soni, Hamza Sajid
Approved By	Luciano Ciattaglia
Website	https://polymesh.network/
Changelog	16/10/2024 - Preliminary Report
Changelog	31/10/2024 - Second Preliminary Report
Changelog	13/11/2024 - Final Report
Platform	Polymesh
Language	Rust
Tags	Substrate, KYC, Decentralized ID
Methodology	https://hackenio.cc/blockchain_methodology

Review Scope

Repository	https://github.com/PolymeshAssociation/polymesh
Commit	e5cc1de8208e6d3461b40b97ed5cf4338ad96010

Audit Summary

The system users should acknowledge all the risks summed up in the risks section of the report

8	7	0	1
Total Findings	Resolved	Accepted	Mitigated

Findings by Severity

Severity	Count
Critical	0
High	0
Medium	3
Low	5

Vulnerability	Severity
F-2024-6030 - Vulnerable and Unmaintained Dependencies	Medium
F-2024-6453 - Denial of Service Risk Due to Oversized PortfolioId Vectors	Medium
F-2024-6616 - Benchmarking Gaps in Contract Instantiation Weight Calculation	Medium
F-2024-6037 - Inconsistent Proposal Handling When Changing Confirmation Requirements	Low
F-2024-6148 - Insufficiencies in Admin Removal Mechanisms	Low
F-2024-6413 - Ticker Linking Mechanism Deficiency	Low
F-2024-6557 - Weight Calculation Neglect of Portfolio Name Length	Low
F-2024-6615 - Insecure Arithmetic Practices Affecting Weight Calculations	Low

Documentation quality

- The protocol documentation is readily accessible on the official Polymesh website.
- The README file provides comprehensive documentation on the build and testing processes.
- The documentation within the codebase is sufficient, offering a solid understanding of the implemented functionalities.
- Both the in-code documentation and the website require updates to align with the latest changes.

Code quality

- The codebase adheres to the highest standards of quality in Rust programming.
- The Substrate code consistently maintains a superior level of quality.
- Benchmarking and weight management are effectively implemented; however, several pitfalls were identified during the audit process.
- Test coverage is recognized as an area requiring improvement.
- The codebase contains unresolved TODO comments, highlighting aspects that necessitate further attention.

Architecture quality

- Polymesh utilizes the mature and widely adopted Substrate framework as the foundational infrastructure for its blockchain.
- The codebase exhibits modularity by effectively organizing functionalities into Substrate pallets.
- Interactions between pallets are concise and thoughtfully designed.
- This modular approach facilitates the safe and efficient implementation of future features.

Table of Contents

System Overview

Polymesh is a Layer 1 blockchain built on the Substrate framework, specifically designed for the issuance and management of security tokens. It integrates various components to achieve its objectives while retaining critical functionalities from the Substrate framework, particularly concerning regulatory compliance requirements like Know Your Customer (KYC) and Customer Due Diligence (CDD).

The audit focused on the changes implemented between Polymesh version 6 and version 7. Key modifications include:

- **Asset Management Enhancement:** Assets are now distinguished by unique identifiers, replacing the previous reliance on Ticker, thereby improving asset management efficiency.
- **Staking Mechanism Updates:** The staking system has been updated to align more closely with the latest Substrate Staking Pallet, enhancing its functionality while preserving essential features unique to Polymesh, such as Customer Due Diligence (CDD).
- **Multi-Signature Proposal Management Enhancements:** Adjustments have been implemented to enhance the management of MultiSignature accounts, including improvements to proposal processes and the streamlining of identity and key-related operations.
- **Support for Mediators in Settlement Processes:** An external party can now be included in the instruction, with their affirmation required for approval.

Risks

- Several issues identified within the audit scope were not included in the report, as the Polymesh team implemented fixes independently. Notable updates include the weights implementation and various migration fixes detailed here and here. These fixes were validated during the audit retesting phase.

Findings

Vulnerability Details

F-2024-6030 - Vulnerable and Unmaintained Dependencies - Medium

Description:

During a security analysis of the Polymesh node using `cargo audit`, several vulnerabilities and maintenance issues were identified across both direct and indirect dependencies. This report outlines significant vulnerabilities and concerns within the codebase.

Direct dependency:

Crate: parity-util-mem

- **Version:** 0.12.0
- **Issues:** `parity-util-mem` is unmaintained
- **Details:** https://rustsec.org/advisories/RUSTSEC-2022-0080

Indirect Dependencies:

Crate: rustls

- **Version:** 0.19.1, 0.20.9
- **Issues:** `rustls::ConnectionCommon::complete_io` could fall into an infinite loop based on network input
- **Details:** RUSTSEC-2024-0336
- **Severity:** 7.5 (High)
- **Solution:** Upgrade to `>= 0.23.5` OR `>= 0.22.4`, `< 0.23.0` OR `>= 0.21.11`, `< 0.22.0`

Crate: webpki

- **Version:** 0.21.4
- **Issues:** webpki: CPU denial of service in certificate path building
- **Details:** RUSTSEC-2023-0052
- **Severity:** 7.5 (High)
- **Solution:** Upgrade to `>= 0.22.2`

Crate: curve25519-dalek

- **Version:** 2.1.3, 3.2.0, 4.1.1
- **Issues:** Timing variability in `curve25519-dalek`'s `Scalar29::sub`/`Scalar52::sub`
- **Details:** RUSTSEC-2024-0344
- **Solution:** Upgrade to `>= 4.1.3`

Crate: ed25519-dalek

- **Version:** 1.0.1
- **Issues:** Double Public Key Signing Function Oracle Attack on `ed25519-dalek`
- **Details:** RUSTSEC-2022-0093
- **Solution:** Upgrade to `>= 2`

Crate: aes-soft

- **Version:** 0.6.4
- **Issues:** `aes-soft` is unmaintained. It has been merged into the `aes` crate
- **Details:** https://rustsec.org/advisories/RUSTSEC-2021-0060
- **Solution:** Please use the `aes` crate going forward.

Crate: aesni

- **Version:** 0.10.0
- **Issues:** `aesni` is unmaintained. It has been merged into the `aes` crate
- **Details:** https://rustsec.org/advisories/RUSTSEC-2021-0059
- **Solution:** Please use the `aes` crate going forward.

Crate: ansi_term

- **Version:** 0.12.1
- **Issues:** `ansi_term` is unmaintained
- **Details:** https://rustsec.org/advisories/RUSTSEC-2021-0139

Crate: mach

- **Version:** 0.3.2
- **Issues:** `mach` is unmaintained
- **Details:** https://rustsec.org/advisories/RUSTSEC-2020-0168

Crate: parity-wasm

- **Version:** 0.45.0
- **Issues:** Crate `parity-wasm` deprecated by the author, wasm-tools is adviced to be used instead.
- **Details:** https://rustsec.org/advisories/RUSTSEC-2022-0061

Crate: proc-macro-error

- **Version:** 1.0.4
- **Issues:** `proc-macro-error` is unmaintained
- **Details:** https://rustsec.org/advisories/RUSTSEC-2024-0370

Crate: serde_cbor

- **Version:** 0.11.2

- **Issues:** `serde_cbor` is unmaintained
- **Details:** https://rustsec.org/advisories/RUSTSEC-2021-0127

Crate: atty

- **Version:** 0.2.14
- **Issues:** Unsound. Potential unaligned read
- **Severity:** Low
- **Details:** https://rustsec.org/advisories/RUSTSEC-2021-0145

Assets:

- Dependencies

Status: Mitigated

Classification

Impact:	4/5
Likelihood:	1/5
Severity:	Medium

Recommendations

Remediation:

The crates, aside from `parity-util-mem`, are not directly used in the current codebase but are dependencies of other packages. Many of these issues stem from outdated pallets or Substrate package versions. It is crucial to identify and update these, along with other outdated dependencies, to the latest versions—even if they are not immediately associated with vulnerable or unmaintained indirect dependencies. The primary goal is to utilize appropriate, updated packages, as outlined below:

- Upgrade `curve25519-dalek` to version `>= 4.1.3`, `ed25519-dalek` to version `>= 2.0`, `rustls` to `>= 0.23.5`, `>= 0.22.4`, or `>= 0.21.11` depending on compatibility, and `webpki` to `>= 0.22.2`.
- Replace or update the following unmaintained crates:
  - `aes-soft` and `aesni`: Merge into the `aes` crate.
  - `ansi_term`, `mach`, `parity-util-mem`, `parity-wasm`, `proc-macro-error`, `serde_cbor`, and `atty`: Replace with actively maintained alternatives.

Additionally, it is recommended to develop and maintain a strategy for managing dependencies to ensure the project consistently uses well-supported and secure libraries:

- Regularly perform security audits on dependencies using `cargo audit` to identify and address potential vulnerabilities.
- Always use the latest stable versions of dependencies.
- If a dependency becomes unmaintained or deprecated, replace it with an actively maintained alternative.
- Adhere to secure coding practices to minimize the risk of introducing security vulnerabilities.

Resolution:	A direct vulnerable dependency has been removed; however, updating other dependencies is not feasible until the project is migrated to a newer Substrate version.

F-2024-6453 - Denial of Service Risk Due to Oversized PortfolioId Vectors - Medium

Description: A denial-of-service (DoS) risk has been identified in the `settlement` pallet, related to the use of unbounded `Vec<PortfolioId>` inputs in several extrinsics, including:

- `affirm_with_receipts`
- `add_and_affirm_instruction`
- `affirm_instruction`
- `withdraw_affirmation`
- `affirm_with_receipts_with_count`
- `affirm_instruction_with_count`
- `withdraw_affirmation_with_count`
- `add_and_affirm_with_mediators`

The core issue arises from the fact that the weight calculations for these extrinsics omit the length of the `Vec<PortfolioId>`. For example, the weight function for `add_and_affirm_instruction` accounts solely for the legs vector, as shown below:

```
fn add_and_affirm_instruction_legs(legs: &[Leg]) -> Weight
```

Furthermore, there are no restrictions on the number of `PortfolioId` entries that can be passed, allowing attackers to submit transactions with excessively large vectors without bearing a corresponding increase in gas costs.

This creates an opportunity for malicious actors to exploit the system by flooding the network with transactions containing oversized vectors, potentially overloading nodes and leading to degraded performance, such as delayed block production or temporary node halts.

Assets:

- settlement

Status: Fixed

Classification

Impact: 3/5

Likelihood: 2/5

Severity: Medium

Recommendations

Remediation: To mitigate this risk, it is recommended to:

- Adjust the weight calculations for the affected extrinsics to properly account for the length of the vector.
- Replace the usage of `Vec` with `frame_support::BoundedVec`, which explicitly limits the maximum number of portfolios that can be provided.

These changes will ensure that the resource consumption is proportional to the input size and prevent potential DoS attacks by limiting the impact of oversized vectors.

F-2024-6616 - Benchmarking Gaps in Contract Instantiation Weight Calculation - Medium

Description: The identified issue pertains to the weight calculation mechanisms for the extrinsics `instantiate_with_code_perms`, `instantiate_with_hash_perms`, `instantiate_with_code_as_primary_key`, and `instantiate_with_hash_as_primary_key` within the `contracts` pallet. These extrinsics accept a `data` parameter, which is subsequently passed to the contract's constructor upon instantiation. An example function definition, along with its corresponding weight function, is shown below:

pallets/contracts/src/lib.rs:463:

```
#[weight = Module::<T>::weight_instantiate_with_code(&code, &salt,
Some(perms)).saturating_add(*gas_limit)]
pub fn instantiate_with_code_perms(
origin,
endowment: Balance,
gas_limit: Weight,
storage_deposit_limit: Option<Balance>,
code: Vec<u8>,
data: Vec<u8>,
salt: Vec<u8>,
perms: Permissions
) -> DispatchResultWithPostInfo
```

Currently, the `data` vectors are not subject to any explicit size constraints. Furthermore, as illustrated in the code snippet above, the weight benchmarking process—responsible for determining dispatchable fee calculations—does not account for the potential impact of the `data` parameter. Instead, benchmarks are conducted with empty vectors, resulting in weight estimations that do not adequately reflect the resource consumption associated with non-empty or large `data` inputs.

This oversight introduces a considerable risk of denial-of-service (DoS) attacks, as malicious actors could exploit the system by submitting extrinsics with excessively large `data` vectors without being subject to proportional fee adjustments. Such misuse could lead to resource exhaustion, adversely affecting node performance or even causing temporary halts.

Assets:

- contracts

Status: Fixed

Classification

Impact: 2/5

Likelihood:	4/5
Severity:	Medium

Recommendations

Remediation: To mitigate this risk, it is recommended to revise the benchmarking and weight calculation procedures to take into account the size of the `data` parameter. This adjustment will help ensure that the calculated weights accurately reflect the resource demands associated with varying input sizes. As a reference, a similar benchmarking implementation can be found in the FRAME's `contracts` pallet here.

Furthermore, it is advisable to establish an upper limit on the `data` vector length to prevent the submission of excessively large inputs, thereby reducing the risk of potential misuse and ensuring the system's stability.

F-2024-6037 - Inconsistent Proposal Handling When Changing Confirmation Requirements - Low

Description: The issue arises in the `change_sigs_required` and `change_sigs_required_via_admin` methods within the `multisig` pallet, due to a logical inconsistency in how the number of required approving votes is managed. Specifically, when the required vote count changes, existing proposals are not automatically reassessed for eligibility for execution or rejection.

Currently, existing proposals are processed based on the updated threshold only when `approve` or `reject` is explicitly called. If the required number of approving votes is reduced, proposals that meet the new criteria for execution remain unprocessed until an additional vote is received. Similarly, if the required number of votes is increased, proposals that should be rejected remain unresolved until further rejecting votes are cast.

While this behavior does not introduce immediate security risks, it can cause unnecessary delays and prevent resolved proposals from being processed. This inconsistency disrupts the proposal lifecycle and undermines the system's overall reliability.

Assets:

- multisig

Status: Fixed

Classification

Impact: 1/5

Likelihood: 4/5

Severity: Low

Recommendations

Remediation: To address the issue, implement logic to automatically reassess all outstanding proposals when the required number of approving votes changes. This reassessment should trigger automatic execution or rejection based on the updated vote thresholds, ensuring that unresolved proposals do not remain stuck.

Alternatively, consider either updating the handling of previously created proposals according to the old threshold or deleting them to avoid confusion regarding the new threshold. The choice between these options should align with the project's overall goals and approaches.

If the current behavior is intentional, document the rationale clearly in both the codebase and external documentation. Consistent and efficient proposal management is crucial for maintaining the system’s smooth operation.

F-2024-6148 - Insufficiencies in Admin Removal Mechanisms - Low

Description: The current implementation of the `multisig` pallet allows for the addition of an admin to a multisig account when consensus is reached among the signers. However, there are notable insufficiencies in the admin removal mechanisms. Currently, signers can only remove the admin by assigning another account to this role. The only way to completely remove the admin, once added, is for the admin account itself to execute the `remove_admin_via_admin` action for its own removal.

While this does not pose an immediate security threat, it creates a risk of excessive centralization of power within the admin account. This becomes problematic if the admin account misbehaves or becomes compromised, as signers have limited options for resolving the issue.

Assets:

- multisig

Status: Fixed

Classification

Impact: 1/5

Likelihood: 4/5

Severity: Low

Recommendations

Remediation: Implement a `remove_admin` method that can be called only by the multisig account itself. This method should facilitate the removal of the admin if one is present. By addressing this issue, you will improve the flexibility and security of the admin management process, mitigating risks associated with centralization and enhancing the resilience of the system.

F-2024-6413 - Ticker Linking Mechanism Deficiency - Low

Description: The issue arises from the implementation of the `link_ticker_to_asset_id` extrinsic in the `asset` pallet. This function establishes a connection between a ticker and an AssetID but fails to validate whether the asset is already linked to an existing ticker. As a result, a new connection is created without properly disconnecting the original ticker. Specifically, the `TickerAssetID` continues to store the initial ticker-AssetID pair, leading to persistent inconsistencies.

This oversight prevents the original ticker from being transferred, re-registered, or linked to another asset, as the `ensure_ticker_not_linked` function fails for the ticker. Consequently, tickers may become frozen, introducing risks of storage discrepancies and potential ticker manipulation.

Assets:

- asset

Status: Fixed

Classification

Impact: 2/5

Likelihood: 2/5

Severity: Low

Recommendations

Remediation:

To address this issue, the following actions are recommended:

- **Validation Check**: Implement a validation step in the `link_ticker_to_asset_id` extrinsic to ensure that the asset is not already linked to a different ticker before establishing a new connection.
- **Unlinking Extrinsic**: Introduce a dedicated extrinsic to enable seamless unlinking of tickers from assets. This extrinsic should properly manage the pallet state and update the expiry time of ticker registrations in the `UniqueTickerRegistration` map following the unlinking process.

Implementing these changes will prevent storage discrepancies, ensure proper ticker management, and mitigate the risk of unexpected ticker freezing.

F-2024-6557 - Weight Calculation Neglect of Portfolio Name Length - Low

Description: The extrinsic responsible for portfolio creation currently calculates weight without factoring in the length of the `PortfolioName`. Specifically, the function definition with its associated weight information is as follows:

```
#[weight = <T as Config>::WeightInfo::create_portfolio()]
pub fn create_portfolio(origin, name: PortfolioName) -> DispatchRes
ult
```

This leads to suboptimal and unfair fee calculations, as varying sizes of `PortfolioName` are treated the same, even though longer names require more storage resources on-chain.

Although the length of the `PortfolioName` vector is capped at 2048 characters, this discrepancy still allows for potential inefficiencies or minor abuse, where larger names incur the same cost as smaller ones, leading to an imbalanced resource usage.

Assets:

- portfolio

Status: Fixed

Classification

Impact: 1/5

Likelihood: 1/5

Severity: Low

Recommendations

Remediation: It is recommended to adjust the weight calculation for the `create_portfolio` extrinsic to include the length of the `PortfolioName`. This change will ensure that fees are proportionate to the actual storage requirements, promoting fairness and preventing potential misuse through excessively long portfolio names.

F-2024-6615 - Insecure Arithmetic Practices Affecting Weight Calculations - Low

Description:

The issue pertains to the use of unsafe arithmetic operations in weight calculation routines, which play a pivotal role in determining appropriate transaction fees. Inaccurate weight calculations can introduce the risk of Denial of Service (DoS) by failing to align fees with the actual resource usage.

Two specific occurrences have been identified:

- In the `utility` pallet, three instances of unsafe arithmetic appear in the weight calculations for the `batch` and `batch_all` functions, identifiable by searching for `base_weight + weight`. Given that the `utility` pallet underpins various functionalities across multiple pallets, any vulnerabilities in its weight calculations pose a systemic risk.
- In the `assets` pallet, the weight calculation for the `create_asset_with_custom_type` extrinsic presents a potential issue. Specifically, the summation of the weights for the `create_asset` and `register_custom_asset_type` could lead to overflow. This overflow would result in a misalignment between the calculated fees and the actual computational and storage resources consumed. The implementation is as follows:

```
#[weight = <T as Config>::WeightInfo::create_asset(
asset_name.len() as u32,
asset_identifiers.len() as u32,
funding_round_name.as_ref().map_or(0, |name| name.len()) as u32
) + <T as Config>::WeightInfo::register_custom_asset_type(custom_as
set_type.len() as u32)]
```

While these scenarios are difficult to exploit and thus present a low probability of occurrence, addressing them is critical to preemptively safeguard against any severe outcomes that may arise from their exploitation.

Assets:

- utility
- asset

Status: Fixed

Classification

Impact: 3/5

Likelihood: 1/5

Severity: Low

Recommendations

Remediation: To address these concerns, it is recommended to replace the current arithmetic operations with `saturated_add` in the identified cases. This will prevent any potential overflow and ensure that weight calculations remain within safe bounds, thereby maintaining appropriate fee determination.

Additionally, it is advisable to perform a review of other similar weight calculation logic across the pallets to identify and address any other instances where unsafe arithmetic could introduce risks.

Observation Details

F-2024-6020 - Chain Specification Error in Local Network Setup - Info

Description:

During the operator simulation of Polymesh nodes, a problem occurred when attempting to simulate a local network. The simulation was initiated using the following steps:

```
cd scripts/cli
npm install
./run.sh
```

An error interrupted the process, which upon investigation, was traced to the `chain` flag in the configuration file. The flag was set to `ci-local` as shown below:

scripts/cli/environment.config.js:2:

```
let chain = "ci-local";
```

This configuration caused the local network initialization to fail due to the absence of a valid chain specification for `ci-local`. The issue was resolved by updating the `environment.config.ts` file and replacing `ci-local` with `local`, enabling the local network to run successfully.

Assets:

- Environment Setup

Status: Fixed

Recommendations

Remediation:

The issue stems from an incorrect setting within the local network configuration. It is crucial to verify that all configuration files are accurately defined and validated prior to executing the application.

In this case, the `chain` flag in the configuration file must point to a valid chain specification. If `ci-local` is invalid or missing, it should be replaced with a valid specification, such as `local`, to ensure proper network initialization.

Addressing this configuration issue streamlines the onboarding process for new developers, facilitating smoother local development and testing.

F-2024-6042 - Proposal Creation Lacks Expiry Time Validation - Info

Description:

The current implementation allows for an optional expiry time when creating a proposal, which is intended to prevent approvals or rejections after this time has passed. However, there is no validation to ensure that the expiry time is set to a future moment. As a result, a proposal can be created with an expiry time that has already elapsed, rendering it immediately invalid. Consequently, any `approve` or `reject` votes on such proposals will result in errors, leading to unnecessary state clutter.

Assets:

- multisig

Status: Fixed

Recommendations

Remediation:

The application should implement a validation check to ensure that the expiry time of a proposal is set to a future date and time. This can be done by comparing the expiry time to the current system time at the moment of proposal creation. If the expiry time is not later than the current time, the system should reject the proposal and return an error message to the user, indicating that the expiry time must be set to a future moment.

Additionally, it would be beneficial to implement a cleanup mechanism that periodically checks for and removes expired proposals, to prevent state clutter. This could be done either on a regular schedule or triggered by certain events, such as the creation of a new proposal.

F-2024-6058 - Code Quality Warnings Highlighted in Static Analysis - Info

Description:

The static analysis of the Polymesh codebase using `cargo clippy` revealed multiple code quality warnings, indicating areas where the code can be refined to improve maintainability, readability, and performance. The findings are categorized below, highlighting the specific issues detected along with their frequency. A complete list of warnings can be generated by running `cargo clippy`.

Several code quality issues were identified, affecting different aspects of the code:

- Needless Borrow 10 occurrence
- Needless Borrows for Generic Args 17 occurrences
- Explicit Auto-Deref 2 occurrences
- Needless Range Loop 3 occurrences
- Suspicious Doc Comments 1 occurrences
- Len Without is_Empty 3 occurrences
- For KV Map 1 occurrences
- Derivable Impl 1 occurrences
- Should Implement Trait 1 occurrences
- Unnecessary Closure Used for Option::None 6 occurrences
- Unused Unit Expression 4 occurrences
- Let Binding with Unit Value 1 occurrences
- Boxed Local 1 occurrences
- From Over Into 2 occurrences
- Unnecessary Cast 2 occurrences
- Too Many Arguments 28 occurrences
- Non-Minimal cfg 1 occurrences
- Collapsible If 1 occurrences
- Redundant Field Names 1 occurrences
- Crate References in Macros 1 occurrences
- Tabs in Doc Comments 11 occurrences
- Type Complexity 3 occurrences

Unsafe Arithmetic:

Warnings flagged by `clippy::arithmetic_side_effects` indicate potential risks in arithmetic operations, such as overflows and underflows. Below is a summary of warnings by module:

- **polymesh-primitive**: 57 warnings
- **polymesh-common-utilities (lib)**: 9 warnings
- **pallet-transaction-payment (lib)**: 6 warnings
- **pallet-balances (lib)**: 22 warnings
- **pallet-identity (lib)**: 3 warnings
- **pallet-multisig (lib)**: 18 warnings

- **pallet-external-agents (lib)**: 5 warnings
- **pallet-committee (lib)**: 4 warnings
- **pallet-portfolio (lib)**: 7 warnings
- **pallet-utility (lib)**: 6 warnings
- **pallet-staking (lib)**: 86 warnings
- **polymesh-contracts (lib)**: 13 warnings
- **pallet-protocol-fee (lib)**: 2 warnings
- **polymesh-runtime-common (lib)**: 9 warnings
- **pallet-compliance-manager (lib)**: 4 warnings
- **pallet-statistics (lib)**: 7 warnings
- **pallet-pips (lib)**: 13 warnings
- **pallet-asset (lib)**: 39 warnings
- **pallet-nft (lib)**: 7 warnings
- **pallet-corporate-actions (lib)**: 21 warnings
- **pallet-settlement (lib)**: 7 warnings
- **pallet-sto (lib)**: 7 warnings
- **polymesh-runtime-mainnet (lib)**: 4 warnings
- **polymesh-runtime-testnet (lib)**: 4 warnings
- **polymesh-runtime-develop (lib)**: 4 warnings
- **polymesh (bin "polymesh")**: 3 warnings

Unsafe Type Casting:

The warnings below were flagged by `clippy::cast_possible_truncation` indicate areas where type casting may lead to data loss or other unintended effects. Below is a breakdown of the number of warnings generated by each module:

- **polymesh-primitives-derive (lib)** generated 1 warning
- **polymesh-primitives (lib)** generated 5 warnings
- **polymesh-common-utilities (lib)** generated 7 warnings
- **pallet-transaction-payment (lib)** generated 2 warnings
- **pallet-balances (lib)** generated 1 warning
- **pallet-identity (lib)** generated 6 warnings
- **pallet-multisig (lib)** generated 6 warnings
- **pallet-committee (lib)** generated 5 warnings
- **pallet-test-utils (lib)** generated 1 warning
- **pallet-external-agents (lib)** generated 4 warnings
- **pallet-portfolio (lib)** generated 1 warning
- **pallet-group (lib)** generated 3 warnings
- **pallet-treasury (lib)** generated 1 warning
- **polymesh-contracts (lib)** generated 8 warnings
- **pallet-staking (lib)** generated 24 warnings
- **pallet-protocol-fee (lib)** generated 1 warning
- **pallet-utility (lib)** generated 16 warnings
- **polymesh-runtime-common (lib)** generated 1 warning
- **pallet-compliance-manager (lib)** generated 3 warnings
- **pallet-statistics (lib)** generated 4 warnings
- **pallet-asset (lib)** generated 16 warnings
- **pallet-nft (lib)** generated 3 warnings

- **pallet-corporate-actions (lib)** generated 7 warnings
- **pallet-settlement (lib)** generated 13 warnings
- **node-rpc (lib)** generated 2 warnings
- **pallet-sto (lib)** generated 1 warning
- **polymesh-runtime-develop (lib)** generated 2 warnings
- **polymesh-runtime-testnet (lib)** generated 2 warnings
- **polymesh-runtime-mainnet (lib)** generated 2 warnings

The analysis reveals opportunities for improvements in coding practices to enhance the overall quality, safety, and performance of the Polymesh codebase. It is recommended to address these findings incrementally, prioritizing changes based on their impact and relevance to the project's goals.

Assets:

- Code Quality

Status: Accepted

Recommendations

Remediation: To enhance code safety and maintainability, the following recommendations should be considered across different aspects of the codebase, applying them where appropriate:

- **Avoid Unnecessary Borrowing**: Utilize values directly instead of applying redundant borrows, simplifying the code and reducing overhead.
- **Leverage Auto-Dereferencing**: Remove explicit dereferencing operations to take advantage of Rust's auto-dereferencing feature, streamlining code execution.
- **Refactor Loops**: Optimize iteration by directly iterating over collections, enhancing clarity and performance.
- **Avoid Unnecessary Casting**: Eliminate redundant type casting operations where the type remains unchanged, maintaining consistency and preventing potential errors.

Unsafe Arithmetic Practices:

- **Utilize Safe Arithmetic Functions**: Where feasible, replace unsafe arithmetic operations (`+`, `-`, `*`, `/`) with corresponding safe functions such as `checked_add`, `checked_sub`, `checked_mul`, and `checked_div` to prevent potential overflow and underflow issues.
- **Handle Errors Explicitly**: In situations where arithmetic operations may potentially fail, incorporate methods that allow for graceful error handling, ensuring predictable behavior and robust code execution.

Unsafe Type Casting Practices:

- **Adopt Safe Conversions**: Use safe conversions like `try_from` instead of direct type casting where there is a risk of truncation or data loss.
- **Account for Architecture Differences**: When relevant, implement safeguards to ensure type conversions are safe across different architectures (32-bit and 64-bit), addressing potential discrepancies in type sizes.
- **Test Type Limits Thoroughly**: Include unit tests to validate type conversions at the boundaries of the type ranges wherever the risk of truncation exists, ensuring reliable casting.
- Applying these improvements where appropriate will help ensure the code remains safe, maintainable, and efficient without imposing unnecessary overhead.

F-2024-6137 - Advised Enhancements to the Test Suite Coverage - Info

Description:

Current test suite analysis raises potential risks to codebase stability and maintainability. Key points:

1. **Inadequate Code Coverage:** Tarpaulin report indicates 58.70% code coverage, leaving 5,540 lines potentially unexercised. This elevates the risk of undetected defects impacting production stability and security.
2. **Pallets Coverage Refinement**: While several pallets require enhancement, particular attention should be given to **contracts**, **permissions**, and **protocol-fee** due to their notably low coverage.
3. **Test Compilation Issues:** Syntax errors in `pallets/identity/src/lib.rs` tests hinder execution, leaving associated code segments untested. This may lead to regressions during future refactoring or enhancements.
4. **Runtime Errors During Testing:** "`BoundedVec` exceeds its limit" errors during testing point towards potential data structure or storage handling issues. These could contribute to application instability or unexpected termination in specific scenarios.
5. **Presence of Ignored Test Cases:** 24 ignored tests suggest potentially unverified functionality, reducing confidence in the system's comprehensive behavior.

The table below contains coverage information for packages and pallets.

packages/pallets	Tested Lines	Total Lines	Coverage (%)
asset	644	1030	62.52%
balances	281	467	60.17%
base	23	28	82.14%
committee	147	202	72.77%
common	29	43	67.44%
common/traits	151	258	58.53%
compliance-manager	213	290	73.45%
contracts	153	516	29.65%
corporate-actions	403	591	68.19%
external-agents	105	172	61.05%
group	210	272	77.21%
identity	606	704	86.08%
multisig	283	469	60.34%
nft	182	301	60.47%
permissions	20	38	52.63%
pips	381	434	87.79%

packages/pallets	Tested Lines	Total Lines	Coverage (%)
portfolio	258	338	76.33%
protocol-fee	44	77	57.14%
relayer	125	167	74.85%
runtime/common	68	289	23.53%
runtime/develop	6	28	21.43%
runtime/extensions	9	23	39.13%
settlement	694	1213	57.22%
staking	1063	1701	62.49%
statistics	171	355	48.17%
sto	153	205	74.63%
sudo	34	47	72.34%
transaction-payment	113	247	45.75%
treasury	34	58	58.62%
utility	115	209	55.02%
primitives	322	560	57.50%
primitives_derive	0	14	0.00%

Impact

- These issues collectively reduce test suite reliability, impacting the ability to comprehensively assess code correctness. Untested code combined with errors during testing increases the likelihood of undiscovered bugs and vulnerabilities.
- Additionally, incomplete testing can complicate code refactoring and maintenance efforts. The absence of thorough test coverage may result in unintended consequences and regressions following code modifications. These factors could affect the project's overall quality, stability, and future development.

Assets:

- Code Quality

Status: Accepted

Recommendations

Remediation: To mitigate the identified risks, the following actions are strongly recommended:

1. **Prioritize Test Development:** Devote dedicated resources to systematically increase test coverage across the codebase, focusing initially on critical modules and edge cases. Employ a combination of unit and integration tests to verify individual components and their interactions.
2. **Investigate and Resolve Errors:**
   - **Compilation Errors:** Correct syntax errors in the `pallets/identity/src/lib.rs` tests to enable their execution.

    - **Runtime Errors:** Thoroughly investigate the "BoundedVec exceeds its limit" errors to identify the root cause and implement appropriate fixes to ensure robust data structure and storage management.
3. **Review and Address Ignored Test Cases:** Re-evaluate ignored test cases to determine their significance. If necessary, enable or rewrite these tests to improve test coverage and ensure critical functionality is verified.
4. **Incorporate Coverage Monitoring:** Integrate code coverage reporting into the development workflow to track progress, identify areas needing attention, and maintain focus on improving test coverage over time.

Resolution:	The Polymesh team has accepted the issue, noting that the code coverage with the old macro approach for Substrate modules is not very accurate.

F-2024-6323 - Enhancing Error Handling Approaches in Pallets Implementations - Info

Description: While the utilization of `unwrap` and `unwrap_err` in Substrate pallets code is discouraged, several instances of their usage have been observed within the codebase. These methods can lead to panics that disrupt the normal operation of the node, potentially causing delays or crashes, thereby compromising the system's stability and reliability. Although acceptable in test cases or benchmarking, their use should be avoided in the Substrate pallets implementation.

The following instances of `unwrap` and `unwrap_err` usage were identified in the codebase, specifically related to the pallets' logic implementation:

- *pallets/balances/src/lib.rs:251*
- *pallets/contracts/src/lib.rs:406*
- *pallets/corporate-actions/src/lib.rs:966*
- *pallets/pips/src/lib.rs:1192*

These occurrences are not proven to be triggerable and do not pose a confirmed security risk; however, they should still be addressed to maintain code quality and robustness.

Assets:

- Code Quality

Status: Fixed

Recommendations

Remediation: To enhance the resilience of the codebase, it is recommended to replace `unwrap` and `unwrap_err` with proper error-handling techniques. Utilize the `Result` and `Option` types where appropriate, or implement concise logic to handle errors without causing a panic. Additionally, consider logging the errors to provide insights into any issues that may arise during execution. Adopting these practices will improve the reliability and safety of the system, ensuring graceful handling of failure scenarios and maintaining stability.

F-2024-6403 - Exploitation Risk in DID Unlinking Caused by Balance Thresholds - Info

Description:

Multi-signature accounts are restricted from unlinking from a Decentralized Identifier (DID) if their balance exceeds a specified threshold (e.g., 1 POLYX). Although this measure aims to prevent accidental loss of funds, it inadvertently introduces a risk.

This leads to adverse consequences, as the legitimate owner loses control over their key, which remains linked to the DID despite their attempts to unlink it. Concurrently, the attacker retains access to the compromised key, thereby enabling potential exploitation for malicious activities by leveraging other vulnerabilities.

Assets:

- multisig

Status: Fixed

Recommendations

Remediation:

It is recommended to remove the threshold restriction that prevents multi-signature accounts from unlinking from a Decentralized Identifier (DID) based on their balance. This change will enhance user control over their keys and mitigate associated risks while still allowing for alternative safeguards against accidental fund loss, such as improved user education and confirmation processes.

F-2024-6412 - Duplicate Function for Token Existence Verification - Info

Description: The codebase includes two distinct functions, `ensure_asset_exists` and `ensure_security_token_exists`, both of which verify the existence of a token associated with a specified `asset_id`. Despite their different names, the overlapping functionality introduces unnecessary duplication in the code.

This redundancy may lead to increased maintenance overhead and the potential for inconsistencies in the future.

Assets:

- Code Quality

Status: Fixed

Recommendations

Remediation: Refactor the code to eliminate redundancy by consolidating the two functions into a single function that verifies the existence of a token associated with a specified `asset_id`. Additionally, it is essential to update all components of the application that previously utilized the original functions to ensure they now leverage the newly refactored function. This will mitigate the risk of potential bugs or inconsistencies.

F-2024-6452 - Instruction Error Reporting Limited to Immediate Errors - Info

Description: The `execute_instruction_report` function within the `settlement` pallet is expected to return all errors encountered during the execution of an instruction. This functionality is exposed through the RPC API to provide detailed feedback on settlement errors.

However, the function currently relies on `ensure_non_expired_affirmations`, which only reports the first error related to an expired affirmation, leaving any subsequent errors underreported. This discrepancy may lead to misunderstandings and hinder the proper evaluation of instruction execution errors.

Similarly, the `ensure_allowed_venue` function, also used in `execute_instruction_report`, returns only the first unauthorized venue it encounters, failing to capture and report any additional unauthorized venues. As a result, the error reporting system may give an incomplete view of the issues affecting the instruction's execution.

These limitations in error reporting could result in a lack of transparency during instruction evaluation and hinder the identification of all necessary corrective actions.

Assets:

- settlement

Status: Accepted

Recommendations

Remediation: To address this issue, consider refactoring the relevant functions, such as `ensure_non_expired_affirmations` and `ensure_allowed_venue`, to return a complete list of all encountered errors instead of just the first one. This approach will provide clearer insights into the reasons for instruction execution failure.

Alternatively, if reporting multiple errors is deemed unnecessary, improve the in-code documentation to more accurately describe the behavior of `execute_instruction_report` and clarify that only the first error is captured. This will help set clearer expectations for developers.

Resolution: The Polymesh team has accepted the issue with the following comment:

> We will leave the implementation as it is for now. Returning additional errors for these cases doesn't seem very useful.

We'll ensure that users are aware of the expected behaviour of these reports.

F-2024-6494 - Inefficiency in Weight Calculation for Mediator Instruction Rejection - Info

Description: The `reject_instruction_as_mediator` function in the `settlement` pallet suffers from an inefficiency in weight calculation:

pallets/settlement/src/lib.rs:871:

```
#[weight = <T as Config>::WeightInfo::reject_instruction_input(None
, true)]
pub fn reject_instruction_as_mediator(
origin,
instruction_id: InstructionId,
number_of_assets: Option<AssetCount>
) -> DispatchResultWithPostInfo {
Self::base_reject_instruction(origin, instruction_id, None, number_
of_assets)
}
```

The optional parameter `number_of_assets` is included to optimize fee calculation by providing the exact number of assets involved. However, the weight function `reject_instruction_input` is currently invoked without passing this parameter, which leads to weight being calculated based on the maximum possible asset count. As a result, the inclusion of `number_of_assets` has no effect, causing unnecessary overestimation of the transaction fee.

Assets:

- settlement

Status: Fixed

Recommendations

Remediation: It is recommended to modify the weight calculation by passing the `number_of_assets` parameter directly to the `reject_instruction_input` function:

```
#[weight = <T as Config>::WeightInfo::reject_instruction_input(numb
er_of_assets, true)]
```

By passing the actual `number_of_assets`, the weight calculation will more accurately reflect the transaction complexity, optimizing both the weight and associated fee for the instruction rejection.

F-2024-6556 - Restrict Self-Addition to AllowedCustodians - Info

Description: This issue highlights a potential improvement in the logical flow of the `allow_identity_to_create_portfolios` extrinsic in `portfolio` pallet. Currently, the extrinsic does not prevent an identity from adding itself to its own `AllowedCustodians` map. While this behavior is not harmful, it is unexpected and counterintuitive.

This could lead to confusion, especially if the identity uses `create_custody_portfolio` to create their own portfolio. In such cases, both the `PortfoliosInCustody` and `PortfolioCustodian` would list the portfolio owner as their own custodian, which contradicts typical expectations.

Assets:

- portfolio

Status: Fixed

Recommendations

Remediation: It is advised to modify the `base_allow_identity_to_create_portfolios` function to ensure that the trusted identity is distinct from the origin's identity. This adjustment will improve the portfolio management logic, preventing confusion or errors as the code develops over time.

This approach will help ensure clarity and consistency in the system's behavior.

Disclaimers

Hacken Disclaimer

The blockchain protocol given for audit has been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in the protocol source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other protocol statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of the blockchain protocol.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Blockchain protocols are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the protocol can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited blockchain protocol.

Appendix 1. Severity Definitions

Severity	Description
Critical	Vulnerabilities that can lead to a complete breakdown of the blockchain network's security, privacy, integrity, or availability fall under this category. They can disrupt the consensus mechanism, enabling a malicious entity to take control of the majority of nodes or facilitate 51% attacks. In addition, issues that could lead to widespread crashing of nodes, leading to a complete breakdown or significant halt of the network, are also considered critical along with issues that can lead to a massive theft of assets. Immediate attention and mitigation are required.
High	High severity vulnerabilities are those that do not immediately risk the complete security or integrity of the network but can cause substantial harm. These are issues that could cause the crashing of several nodes, leading to temporary disruption of the network, or could manipulate the consensus mechanism to a certain extent, but not enough to execute a 51% attack. Partial breaches of privacy, unauthorized but limited access to sensitive information, and affecting the reliable execution of smart contracts also fall under this category.
Medium	Medium severity vulnerabilities could negatively affect the blockchain protocol but are usually not capable of causing catastrophic damage. These could include vulnerabilities that allow minor breaches of user privacy, can slow down transaction processing, or can lead to relatively small financial losses. It may be possible to exploit these vulnerabilities under specific circumstances, or they may require a high level of access to exploit effectively.
Low	Low severity vulnerabilities are minor flaws in the blockchain protocol that might not have a direct impact on security but could cause minor inefficiencies in transaction processing or slight delays in block propagation. They might include vulnerabilities that allow attackers to cause nuisance-level disruptions or are only exploitable under extremely rare and specific conditions. These vulnerabilities should be corrected but do not represent an immediate threat to the system.

Appendix 2. Scope

The scope of the project includes the following components from the provided repository:

Scope Details	
Repository	https://github.com/PolymeshAssociation/polymesh
Commit	e5cc1de8208e6d3461b40b97ed5cf4338ad96010

Components in Scope

The audit encompasses the entire Polymesh repository, with a particular emphasis on analyzing the modifications between versions 7 and 6 of Polymesh.