HACKEN

POLKADEX SECURITY ANALYSIS

Hacken OÜ
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia
support@hacken.io

Intro

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.

The report can be disclosed publicly after prior consent by another party. Any subsequent publication of this report shall be without mandatory consent.

Name	Polkadex
Website	https://polkadex.trade/
Repository	https://github.com/Polkadex-Substrate/Polkadex
Commit	1dd8eed88b5a5181d33b4d83f16ed5bcf559528e
Platform	L1
Network	Polkadex
Languages	Rust
Audit Methodology	Blockchain Protocol and Security Analysis Methodology
Auditor	s.akermoun@hacken.io n.lipartiia@hacken.io y.bratashchuk@hacken.io
Approver	l.ciattaglia@hacken.io
Timeline	17.04.2023 - 22.05.2023 (Preliminary Report)
TimeLine	17.10.2023 (Final Report)

Hacken OÜ
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia
support@hacken.io

Table of contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia
support@hacken.io

Summary

Polkadex is a peer-to-peer cryptocurrency exchange operating within the decentralized finance (DeFi) ecosystem. It employs the Substrate framework, aiming to provide a secure and scalable trading platform with an emphasis on performance.

A key feature of Polkadex is its decentralized orderbook, managed by off-chain workers under governance regulations. This setup allows the exchange to operate in a decentralized manner while still maintaining the functionality of a conventional orderbook.

Polkadex integrates with a broad range of assets from different blockchains through its decentralized Thea bridge technology, which is also managed by governance. This integration facilitates efficient and reliable asset transfers and communication within the platform.

Additionally, Polkadex has a link to the Polkadot ecosystem through a parachain. This connection allows Polkadex to utilize the resources and features of the Polkadot network, further enhancing its capabilities and interoperability in the DeFi sector.

Documentation quality

In the course of enhancing their codebase, the Polkadex development team has made concerted efforts to standardize documentation across both legacy and newly implemented components. They have also adopted improved Rust docstrings, thereby bolstering the overall robustness and readability of the system's code. Notwithstanding these improvements, there remain specific areas that would benefit from additional refinement.

With respect to the granularity of the code documentation, it is noteworthy that the majority of storage values, extrinsics, errors, and events are comprehensively documented. The enhanced Rust docstrings further contribute to this level of detail, facilitating a nuanced understanding of potential error states and system events, thereby contributing to a more robust interpretation of the system's operational mechanics.

However, during the audit, it was observed that high-level architectural documentation for comprehensive system components was conspicuously absent. This necessitated a significant investment of time on the part of the auditing team to independently interpret the system architecture. Subsequent to this, a GitBook was furnished towards the end of the audit, which substantially enhanced our conceptual and high-level understanding of the system.

The total Documentation Quality score is **7** out of 10.

Code quality

The Polkadex Layer 1 project demonstrates commendable practices in Rust programming. Running the linter with its default configuration does not show any warnings related to flagrant idiomatic Rust issues. This adherence to Rust best practices indicates a focus on maintainability, readability, and stability within the codebase.

All pallets are covered by tests, and sensitive functions within those pallets have also been specifically addressed through targeted testing.

The Polkadex development team has made significant advancements in addressing previously identified gaps in the benchmarking of pallets. Utilizing Substrate's framework, they have enhanced the computational weight calculations for extrinsics, ensuring a more accurate representation of system performance.

During the code review phase of the audit, a substantial number of TODO comments indicated that the codebase might still need to be completed. This raises concerns as these markers represent crucial yet unimplemented logic sections, which could impact the stability and security of the overall system.

The codebase also exhibits usage of unsafe arithmetic operations, which can lead to unexpected behavior or vulnerabilities if not managed properly. Employing safe alternatives where possible is recommended to avoid potential arithmetic issues such as overflow and underflow.

The total Code Quality score is **8** out of 10.

Architecture quality

The Polkadex Layer 1 project has a well-designed architecture that separates off-chain activities from on-chain settlements. This separation ensures that the platform is scalable, secure, and flexible in its design.

The Layer 1 infrastructure settles what happens off-chain, including the operations of the orderbook relayers and Thea bridge workers. This layered approach helps to maintain a clean and manageable codebase, making the system easier to maintain, update, and extend in the future.

An important aspect of the project's architecture is its effective utilization of threshold cryptography in the settlement process. This methodology ensures that transactions are validated through a threshold of signatures from off-chain participants, enhancing the security of the platform. Threshold cryptography brings an additional layer of protection by reducing the risk of key compromise through distributing pieces of the private key among multiple participants. This cryptographic practice enhances the overall security of the system, adding robustness against attacks and system failures.

The project leverages a governance mechanism to manage its decentralized operations. This approach helps to ensure that the platform remains decentralized and trustless, further bolstering its security and aligning with the principles of decentralized finance. By integrating governance into the system architecture, Polkadex provides a framework for collective decision-making, facilitating adaptability and resilience in the platform's evolution.

The sudo pallet is currently in use but will be removed by the team.

The architecture quality score is **9** out of 10.

Security Level

In our analysis of Polkadex, we initially identified a number of security issues requiring attention, which included 1 High, 2 Medium, and 1 Low severity issues. As of now, all of these have been resolved except for one medium severity issue. According to the development team, this remaining issue has been mitigated and does not pose a risk.

The security score is **10** out of 10.

Total score

Considering all metrics, the total score of the report is **9.4** out of 10.

Findings count and definitions

Severity	Findings	Severity Definition
Critical	0	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	1	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	2	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	1	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.
Total	4	

Hacken OÜ
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia
support@hacken.io

Scope of the audit

Protocol Audit

Substrate fork review

- Review of all code changes and missing updates since Substrate clone date

Cryptography and Keys

- Cryptography Libraries
- Keys Generation
- Keystore storage
- Asymmetric (Signing and Verification)

Substrate client configuration review

- Genesis review
- Consensus
- Substrate FRAME pallets usage review
- Standard attacks review (replay, malleability,...)

XCM

- XCM Implementation
- Protocol-level vulnerabilities
- Interoperability vulnerabilities
- Integration vulnerabilities

Runtime & Pallets

- Runtime implementation review
- Pallet asset-handler review
- Pallet thea review
- Pallet chainbridge review
- Pallet liquidity review
- Pallet ocex review
- Pallet pdex-migration review
- Pallet rewards review
- Pallet router review
- Pallet support review
- Pallet swap review
- Pallet ocex review
- Attack scenarios analysis (Weight, race, stack, DoS, state implosion, access control bypass...)

RPC

- RPC implementation review

- Attack scenarios analysis (defaults,DoS, overflows, ..)

Protocol Tests

Node Tests

- Environment Setup
- E2E sync tests
- Consensus tests
- E2E transaction tests

Runtime Tests

- Fuzz tests

Hacken OÜ
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia
support@hacken.io

Issues

Unhandled Error in FeesCollected Insertion Leading to Potential Node Crash

The `pallet-ocex-lmp` pallet experiences a node crash due to an unhandled error resulting from an unchecked conversion from an unlimited vec to a limited `BoundedVec`.

ID	PDX-110
Scope	pallet-ocex-lmp Snapshot Processing
Severity	HIGH
Vulnerability Type	Error handling / Data Validation
Status	Fixed

Description

Within the `pallet-ocex-lmp pallet`, snapshots submitted via `submit_snapshot` are processed, and the accumulated fees are stored in the `FeesCollected` storage item. However, an issue arises when the `FeesCollected` attempts to map each snapshot ID to a list of all collectable fees, stored in a `BoundedVec` with a maximum size of `AssetsLimit`.
pallets/ocex/src/lib.rs:1382:

```
// Fees collected
#[pallet::storage]
#[pallet::getter(fn fees_collected)]
pub(super) type FeesCollected<T: Config> =
    StorageMap<_, Blake2_128Concat, u64, BoundedVec<Fees, AssetsLimit>, ValueQuery>;
```

The size limit for `AssetsLimit` is defined to be 1000:
primitives/polkadex/src/lib.rs:97:

```
#[derive(Debug, Clone, Copy, PartialEq, TypeInfo, Encode, Decode)]
#[cfg_attr(feature = "std", derive(Serialize, Deserialize))]
pub struct AssetsLimit;
impl Get<u32> for AssetsLimit {
    fn get() -> u32 {
        1000
    }
}
```

Each snapshot ID can, therefore, only have a maximum of 1000 `Fees` elements. The `submit_snapshot` implementation, does not check the size of the number of withdrawals to process, which directly affects the number of fee elements. This results in an unexpected node crash when the `BoundedVec::try_from(working_summary.get_fees())` conversion fails due to size constraints, and `unwrap()` is called.
pallets/ocex/src/lib.rs:1043:

```
<FeesCollected<T>>::insert(
    working_summary.snapshot_id,
    BoundedVec::try_from(working_summary.get_fees()).unwrap(),
);
```

Despite size verification taking place in the off-chain worker before snapshot submission, it is crucial to handle potential errors within the L1 blockchain extrinsic. This ensures robustness and resilience of on-chain code against any unexpected or malformed inputs.

Proof of concept

This proof of concept effectively demonstrates a potential node crash due to an unhandled panic within the `submit_snapshot` function of the `pallet-ocex-lmp` pallet. The function is designed to process a snapshot summary of account withdrawals. However, it does not handle the scenario where the size of the withdrawals vector in the snapshot summary exceeds the maximum allowed size, defined by `AssetsLimit`.
This could potentially allow an attacker to submit a snapshot with an excessively large number of withdrawals, causing the chain to panic due to the unhandled error.
"This Proof of Concept (PoC) can be utilized as part of the unit test suite for the `pallet-ocex-lmp pallet`.

```
/// This unit test is designed as a proof-of-concept (PoC) to demonstrate a potential vulnerability
/// in the `submit_snapshot` function of the pallet-ocex-lmp pallet.
/// The `submit_snapshot` function is used to submit a snapshot summary of account withdrawals. However,
/// it doesn't check whether the size of the withdrawals vector in the snapshot summary exceeds the
/// maximum allowed size (AssetsLimit). This could lead to a potential node crash due to an
/// unhandled panic where a sender could submit a snapshot with an extremely large number of withdrawals,
/// which could cause the chain to panic or significantly slow down due to excessive processing time.
#[test]
#[should_panic]
fn test_submit_snapshot_panic() {
    // Initialize test environment
    let _account_id = create_account_id();
    let mut t = new_test_ext();
    t.execute_with(|| {
        // Create a dummy snapshot with 1001 withdrawals.
        // This number is arbitrarily chosen to exceed the limit of maximum allowed assets,
        // thereby triggering a panic when the snapshot is submitted.
        let (snapshot, _public) = get_dummy_snapshot(1001);
        // Attempt to submit the snapshot. This should cause a panic because the number of
        // withdrawals in the snapshot exceeds the maximum allowed limit (AssetsLimit).
        _ = OCEX::submit_snapshot(RuntimeOrigin::none(), snapshot.clone());
    })
}
```

The test passes if it panics as defined by the `#[should_panic]` attribute of the test:

```
% cargo test -p pallet-ocex-lmp test_submit_snapshot_panic
running 1 test
test tests::test_submit_snapshot_panic - should panic ... ok
test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured; 64 filtered out; finished in 0.10s
```

Recommendation

While it's important to have pre-checks in off-chain workers to validate the size of a snapshot before submission, these cannot be solely relied upon for the security of the on-chain extrinsic `submit_snapshot`. It's critical to note that off-chain workers operate in an environment with weaker trust assumptions, and are more prone to errors, manipulations, or even malicious attacks.

Therefore, it's strongly recommended that an additional layer of validation checks is introduced in the on-chain `submit_snapshot` function to ensure robustness and resilience against any unexpected or malformed inputs. This can be achieved by checking the size of the withdrawal vector against the maximum limit `AssetsLimit` before attempting the conversion to a `BoundedVec`.

If the withdrawal vector size exceeds `AssetsLimit`, the function should return an error and stop processing, thus avoiding potential node crashes or excessive processing times. This would also help to mitigate potential Denial-of-Service (DoS) attacks that could be launched by submitting snapshots with an exceedingly large number of withdrawals.

By ensuring the on-chain code is robust against unexpected inputs, we can help to secure the blockchain against potential threats and maintain the integrity and performance of the network.

Hacken OÜ
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia
support@hacken.io

Inaccurate Signature Threshold Calculation

ID	PDX-109
Scope	pallet-ocex-lmp Snapshot Processing
Severity	**MEDIUM**
Vulnerability Type	Arithmetic Errors
Status	Fixed

Description

In the `pallet-ocex-lmp` pallet, new submitted snapshots are processed based on a signature threshold, which is intended to follow a 2/3 majority. However, the current implementation of the signature threshold calculation in the `submit_snapshot` function uses a "round down" approach, which can lead to a lower signature threshold than intended.

As a result, in some cases, the actual signature threshold is below the desired 2/3 majority (**66.67%**). This issue occurs because the floor division used in the code does not round up the resulting value, effectively rounding down the threshold and potentially making it easier for a snapshot to be processed.

The code snippet responsible for this calculation is: *pallets/ocex/src/lib.rs:999*:

```
// Check if we have enough signatures
let total_validators = <Authorities<T>>::get().len();
if working_summary.signed_auth_indexes().len() >=
    total_validators.saturating_mul(2).saturating_div(3)
{
    /* Process snapshot */
}
```

Here, `total_validators` is multiplied by 2 and then divided by 3. However, because both values are `usize`, the division operation will round down the result, which can lead to a lower signature threshold in certain cases.

Recommendation

To address this issue, we recommend updating the signature threshold calculation in the `submit_snapshot` function to use a "round up" approach, ensuring that the actual signature threshold remains at or above the intended 2/3 majority (66.67%).

A possible implementation using the `ceil` function from the `f64` type in Rust is as follows:

```
if working_summary.signed_auth_indexes().len() >=
    ((total_validators as f64 * 2.0 / 3.0).ceil() as usize)
{
    /* Process snapshot */
}
```

By converting the `total_validators` to an `f64` and using the `ceil` function, the calculation will round up the result, ensuring that the signature threshold remains at or above the desired 2/3 majority.

After implementing this change, the signature threshold should be closer to the intended value for for all validator counts, providing a more robust snapshot processing mechanism that adheres to the desired 2/3 majority.

Hacken OÜ
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia
support@hacken.io

Vector of unlimited size in thea-executor

The `ApprovedDeposits` storage value is currently using a `Vec` data structure, which does not have a limit on its size. This opens up the possibility for the vector to grow infinitely.

ID	PDX-116
Scope	thea-executor deposit handling
Severity	**MEDIUM**
Vulnerability Type	Memory exhaustion / DoS
Status	Mitigated

Details

The `thea-executor` pallet has a storage value called `ApprovedDeposits`, which is defined as following: *pallets/thea-executor/src/lib.rs:81*:

```
#[pallet::storage]
#[pallet::getter(fn get_approved_deposits)]
pub(super) type ApprovedDeposits<T: Config> =
    StorageMap<_, Blake2_128Concat, T::AccountId, Vec<Deposit<T::AccountId>>, ValueQuery>;
```

The issue with an unlimited vector is that it can grow infinitely and consume an excessive amount of storage space, potentially causing memory exhaustion.

The issue occurs in the `do_deposit` function, where deposits are pushed into the vector without any checks on its length: *pallets/thea-executor/src/lib.rs:313*:

```
for deposit in deposits {
    <ApprovedDeposits<T>>::mutate(&deposit.recipient, |pending_deposits| {
        pending_deposits.push(deposit.clone())
    })
}
```

This method is then used in both the `incoming_message` function of the `thea` and `thea-message-handler` pallets, that don't perform any checks either.

Although there is a mechanism for deleting elements in the vector in the `claim_deposit` method, it can only be called by the corresponding user and is not an effective way to regulate the length of the vector.

Recommendation

To address the issue of an unlimited vector in the `ApprovedDeposits` storage value, it is recommended to implement a maximum length for the approved deposit and use a data structure that is bounded in size, such as `BoundedVec`:

```
pub(super) type ApprovedDeposits<T: Config> =
    StorageMap<_, Blake2_128Concat, T::AccountId, BoundedVec<Deposit<T::AccountId>, T::DepositsLimit>, ValueQuery>;
```

This will prevent the vector from growing indefinitely and exceeding storage limits.

By implementing a maximum length and using a bounded data structure, the project can help mitigate the risk of memory overflows and DoS attacks.

Hacken OÜ
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia
support@hacken.io

Polkadex Node Fails to Execute

Polkadex node cannot execute with the current Rust toolchain configuration.

ID	PDX-102
Scope	Polkadex Node
Severity	**LOW**
Vulnerability Type	Misconfiguration
Status	Fixed

Description

The Rust toolchain used for building the project is configured in the *rust-toolchain.toml* file to use the latest nightly channel: *rust-toolchain.toml*:

```
[toolchain]
channel = "nightly"
components = [ "rustfmt", "clippy" ]
targets = [ "wasm32-unknown-unknown" ]
```

This configuration is confirmed by the following command output:

```
% cargo --version
cargo 1.71.0-nightly (d0a4cbcee 2023-04-16)
```

Although the project compiles successfully, the Polkadex client fails to launch, and the node shuts down with the following output:

```
$ ./target/release/polkadex-node --dev
2023-04-17 14:38:19 Polkadex Node
2023-04-17 14:38:19 ✌  version 4.0.0-ae0e998d-aarch64-macos
2023-04-17 14:38:19 ❤  by Polkadex OÜ <https://polkadex.trade>, 2017-2023
2023-04-17 14:38:19   Chain specification: Development
2023-04-17 14:38:19    Node name: panoramic-circle-4949
2023-04-17 14:38:19   Role: AUTHORITY
2023-04-17 14:38:19   Database: RocksDb at /var/folders/p7/q2z_t8pj49v8gc4ghk4t0kk00000gn/T/substratewCOYD4/chains/dev/d
2023-04-17 14:38:19    Native runtime: node-282 (polkadex-official-0.tx2.au10)
Error: Service(Client(VersionInvalid("cannot deserialize module: UnknownOpcode(192)")))
2023-04-17 14:38:19 [0]   generated 1 npos voters, 1 from validators and 0 nominators
2023-04-17 14:38:19 [0]   generated 1 npos targets
2023-04-17 14:38:19 Cannot create a runtime error=Other("cannot deserialize module: UnknownOpcode(192)")
```

This issue is known in Parity, as confirmed by issue #13636 and addressed in pull request #13804.

With newer versions of LLVM, some WebAssembly features are silently enabled. The fix in the PR disables the `sign-ext` feature with opcode 192 (0xC0).
The Rust nightly versions that break the runs are `nightly-2023-03-19` and all subsequent versions.

Recommendation

We recommend setting a nightly version that does not use the new LLVM features prior to `nightly-2023-03-19`.
Actually from our test `nightly-2023-03-18` works fine. *rust-toolchain.toml*:

```
[toolchain]
channel = "nightly-2023-03-18"
```

```
components = [ "rustfmt", "clippy" ]
targets = [ "wasm32-unknown-unknown" ]
```

The fix applied by #13804 is currently only available in the master branch of the Substrate repository and not in any new Polkadot branches or monthly releases.
Monitor the Substrate node repository for official releases and apply the updates accordingly when available.

An alternative fix involves applying the changes described in #13804, which consists of modifying the Rust compilation flags for the WebAssembly builder.

Compilation and Linter Warnings

Polkadex L1 Substrate chain has a clean build with a minor single dependency warning.

ID	PDX-101
Scope	Code Quality
Status	Fixed

Description

During the security audit of Polkadex L1 substrate chain, we have compiled the project and checked for linter warnings using `cargo check` and `cargo clippy` with default configuration. The project builds without errors or warnings related to Rust code best practices, indicating a satisfactory level of code quality and Rust code maturity.

However, a minor issue should be addressed in future updates. The `nalgebra` dependency within `linregress`, a dependency of the Substrate `benchmarking` FRAME pallet, generates the following warning:

```
warning: the following packages contain code that will be rejected by a future version of Rust: nalgebra v0.27.1
```

In Substrate repository, a pull request that resolves this warning can be found at #13310.
The proposed changes involve upgrading the `linregress` dependency of the `benchmarking` FRAME pallet from version **0.4.4** to **0.5.1** and updating the relevant syntax in *frame/benchmarking/src/analysis.rs* to match changes in `linregress` public API.
This patch has been applied to the `polkadot-v0.9.39` branch.

This issue is informational and serves as a reminder to address this dependency warning in future updates. Ignoring the warning and not addressing the `nalgebra` dependency issue may lead to compatibility problems with future versions of Rust. This could result in build errors, deprecated code usage, or even security vulnerabilities as the project may no longer receive important updates and fixes from the `nalgebra` library. It is crucial to maintain compatibility with the latest Rust version to ensure the stability and security of the Polkadex L1 Substrate chain.

As the issue PDX-102 was resolved by using an older toolchain instead of the latest nightly, the warning may not appear at the moment. However, it remains an underlying concern and may resurface during future toolchain updates for Polkadex. Therefore, the above recommendations should still be taken into consideration.

Recommendation

We recommend that the Polkadex development team consider upgrading the substrate dependencies to at least `branch = "polkadot-v0.9.39"` to maintain compatibility with future Rust versions. This upgrade will help address the `nalgebra` dependency issue and ensure that the project remains compatible with future updates to Rust and its associated libraries.

As this issue is informational and does not require immediate action, it serves to inform the development team about potential risks associated with the `nalgebra` dependency and encourage proactive planning for future updates.

Additionally, it is crucial to keep dependencies up-to-date and regularly monitor linter warnings to identify and address potential code quality or security issues promptly.

Employment of Sudo Pallet

The sudo FRAME pallet is currently leveraged as an alternative to the governance mechanism.

ID	PDX-115
Scope	Code Quality / Decentralization
Status	Acknowledged (Team will remove sudo pallet after launch)

Description

The runtime configuration incorporates the `sudo-pallet`. *runtime/src/lib.rs:1441*:

```
construct_runtime!(
    pub enum Runtime where
        Block = Block,
        NodeBlock = polkadex_primitives::Block,
        UncheckedExtrinsic = UncheckedExtrinsic
    {
        /* ... */
        Sudo: pallet_sudo::{Pallet, Call, Config<T>, Storage, Event<T>} = 17,
        /* ... */
    }
);
```

The root account, initially set at genesis, is defined as follows:
node/src/chain_spec.rs:318:

```
let root_key = hex!["70a5f4e786b47baf52d5a34742bb8312139cfe1c747fbeb3912c197d38c53332"].into();
```

The Polkadex protocol administration is conducted by governance, which can configure the protocols by invoking extrinsics from pallets. However, within the Runtime implementation of these pallets, it is apparent that the root user could potentially sidestep the governance system, thus serving as an alternative.

Several implementations illustrate this, including the `asset-handler`, `pallet-ocex-lmp`, `chainbridge`, and `liquidity` pallets in the Runtime.
Additionally, the pallets `thea`, `thea-executor`, and `thea-message-handler` still mandate the origin of extrinsics call to be the root user.

Another point of contention is the competitive advantage the root account possesses in collecting fees and redirecting them to an owned address by invoking the `collect_fees` extrinsic from the orderbook pallet, thereby circumventing the democracy mechanism.

Recommendation

1. It is paramount to document comprehensively the projected use of the sudo pallet and the root account within the project. Guarantee that both the development team and end-users are adequately informed about the potential risks and limitations associated with its usage. If plans exist to deactivate the sudo functionality post-network launch, this process should be elaborately documented, akin to the sudo removal outlined by Polkadot.
2. Prior to the removal of the sudo pallet, ensure the regular auditing and monitoring of the sudo pallet and the root account to verify they are not misused or compromising the system's security.

The current implementation does not pose an immediate security risk; however, it does raise potential centralization concerns and is a less desirable design choice. If the root account's private key were to be compromised, this issue could potentially escalate into a vulnerability, leading to unauthorized access to privileged actions.

Inconsistent and Misleading Comments and Logs in Thea Codebase

The codebase for Thea contains misleading comments and logs copied from Orderbook client codebase, which can lead to confusion and misinterpretation.

ID	PDX-120
Scope	Code Quality
Status	Fixed

Description

The issue lies in the fact that comments and logs, which are essential for understanding the operation of the code, are copied from Orderbook clientcodebase. This may lead to confusion, as these comments and logs could be mistakenly interpreted as belonging to the Orderbook worker, rather than the Thea worker.

Recommendation

To enhance the readability and accuracy of the code, it is recommended to revise and update the comments and logs in the Thea codebase. Ensure that each comment and log message accurately reflects the operation of the specific code block to which it pertains.

Incorrect Use of Bitwise OR Operator

This report highlights an issue in the code where the bitwise OR operator is used instead of the logical OR operator, potentially leading to confusion and future maintenance issues.

ID	PDX-108
Scope	Code Quality
Status	Fixed

Description

In the `do_withdraw` function within the `thea-executor` pallet, the following line of code uses the bitwise OR operator (|) instead of the logical OR operator (||): *pallets/thea-executor/src/lib.rs:345*:

```
if pending_withdrawals.is_full() | pay_for_remaining { /* ... */ }
```

While the bitwise OR operator does not cause incorrect behavior in this particular case, it is not idiomatic and can lead to confusion and potential issues when the code is modified in the future.

Recommendation

Replace the bitwise OR operator (|) with the logical OR operator (||) to improve code readability and maintainability:

```
if pending_withdrawals.is_full() || pay_for_remaining { /* ... */ }
```

By making this change, the code will be more in line with standard Rust idioms and less prone to issues caused by misunderstandings or changes in the future.

Magic Numbers As Constants in pallets

Usage of magic numbers as constant, directly hardcoded in pallets code, can lead to maintainability, readability, and potential security issues.

ID	PDX-105
Scope	Code Quality
Status	Fixed

Description

Hardcoding values in the code makes it harder to maintain and can introduce bugs, as the same value may need to be updated in multiple places. It may also make the code less readable, as the meaning behind the hardcoded values might not be immediately clear to other developers working on the project.

In addition, hardcoded values can lead to security vulnerabilities if these values are related to security-sensitive parameters, such as timeouts, limits, or key sizes. An attacker may exploit these vulnerabilities by taking advantage of hardcoded limits or other constraints in the system.

Here is a list of all usage of hardcoded values in the project:

pallets/asset-handler/src/lib.rs:98:

```
impl Get<u32> for WithdrawalLimit {
    fn get() -> u32 {
        5 // TODO: Arbitrary value
    }
}
```

pallets/asset-handler/src/lib.rs:106:

```
impl Get<u32> for AllowlistedTokenLimit {
    fn get() -> u32 {
        50 // TODO: Arbitrary value
    }
}
```

pallets/asset-handler/src/lib.rs:193:

```
#[pallet::storage]
#[pallet::getter(fn get_thea_assets)]
pub type TheaAssets<T: Config> =
    StorageMap<_, Blake2_128Concat, u128, (u8, u8, BoundedVec<u8, ConstU32<1000>>), ValueQuery>;
```

pallets/ocex/src/lib.rs:127:

```
impl Get<u32> for AllowlistedTokenLimit {
    fn get() -> u32 {
        50 // TODO: Arbitrary value
    }
}
```

pallets/thea-executor/src/lib.rs:247:

```
ensure!(beneficiary.len() <= 1000, Error::<T>::BeneficiaryTooLong);
```

Recommendation

To mitigate these issues and improve the code quality, we recommend to move configurable constants to the pallet configuration, allowing them to be adjusted at runtime. this provides flexibility and allows for easier tuning of the system's behavior.

Missing Benchmark Calculations

The project currently lacks benchmark calculations for the extrinsic weights of several pallets. Additionally, some hooks are returning incorrect weights.

ID	PDX-111
Scope	Code Quality / Performance
Status	Fixed

Details

Accurate weight calculations are a critical aspect of maintaining stable and efficient runtime. However, some pallets assign default values for weights. The pallets with missing benchmark calculations are:

- *liquidity*
- *pdex-migration*
- *router*
- *thea*
- *thea-executor*
- *thea-message-handler*

All these pallets use `#[pallet::weight(Weight::default())]` instead. It returns zero values, which can lead to inaccurate transaction fee calculations, imbalanced resource consumption and even vulnerabilities or attack vectors, such as denial-of-service (DoS) attacks.

There are also issues with extrinsics of the *chainbridge* and *ocex* pallets. The *chainbridge* pallet provides a default weight of `#[pallet::weight(195_000_000)]` for most of its functions. Meanwhile, the *ocex* pallet's `whitelist_orderbook_operator` method has a constant weight of `#[pallet::weight(10000)]` without any accompanying comments or computations. This approach is less desirable compared to automatically generated weights.

Another issue worth noting is the default weights that are returned by `on_initialize` hooks:

pallets/chainbridge/src/lib.rs:260:

```
#[pallet::hooks]
impl<T: Config> Hooks<T::BlockNumber> for Pallet<T> {
    fn on_initialize(_n: T::BlockNumber) -> Weight {
        // Clear all bridge transfer data
        BridgeEvents::<T>::kill();
        Weight::default() // TODO: This is not zero
    }
}
```

pallets/thea-executor/src/lib.rs:138:

```
#[pallet::hooks]
impl<T: Config> Hooks<BlockNumberFor<T>> for Pallet<T> {
    fn on_initialize(block_no: T::BlockNumber) -> Weight {
        /* ... */
        //TODO: Clean Storage
        Weight::default()
```

```
    }
}
```

Recommendation

To ensure that the project has a performant and stable runtime environment, it is recommended to include benchmark calculations for all hooks and extrinsics that do not currently have them. This can be done by following these steps:

- Identify all extrinsics within the project's pallets and group them accordingly.
- Set up a benchmarking framework, such as `frame-benchmarking`, to automate the process of calculating extrinsic weights.
- Create benchmarking tests for each extrinsic, focusing on various input parameters and potential edge cases.
- Run the benchmark tests on a range of hardware configurations to account for varying performance capabilities of potential nodes.
- Analyze the results to determine accurate weight values for each extrinsic, considering the expected usage patterns and resource consumption.
- Integrate the calculated extrinsic weights into the project's pallets and runtime configuration.
- Regularly review and update the benchmark calculations as the project evolves to maintain performance and efficiency.

By implementing benchmark calculations for extrinsic weights, you will ensure a more efficient and stable runtime environment, leading to optimized resource usage, accurate transaction fees, and overall improved performance in your project.

Non-Idiomatic Error Handling in `collect_fees` Function of `pallet-ocex-lmp` Pallet

ID	PDX-118
Scope	Code Quality
Status	Fixed

Details

The error handling approach in the `collect_fees` function of the `pallet-ocex-lmp` pallet is not idiomatic in Rust. The current implementation uses `unwrap_or_default` after a `try_push` operation, which is not the best practice. This approach may ignore potential errors in the event of a failure and could complicate debugging.

The code in question is found 2 times within the `collect_fees` function: *pallets/ocex/src/lib.rs:786-791*:

```
internal_vector.try_push(fees).unwrap_or_default();
```

Recommendation

A more idiomatic approach in Rust is to use `if let` to match the `Result` returned by the `try_push` operation, allowing for immediate and clear error handling. The suggested refactor is as follows:

```
/// Withdraws Fees Collected
///
/// params:  snapshot_number: u32
#[pallet::call_index(11)]
#[pallet::weight(<T as Config>::WeightInfo::collect_fees(1))]
pub fn collect_fees(
    origin: OriginFor<T>,
    snapshot_id: u64,
    beneficiary: T::AccountId,
) -> DispatchResult {
    // TODO: The caller should be of operational council
    T::GovernanceOrigin::ensure_origin(origin)?;
```

```
    ensure!(
        <FeesCollected<T>>::mutate(snapshot_id, |internal_vector| {
            while internal_vector.len() > 0 {
                if let Some(fees) = internal_vector.pop() {
                    if let Some(converted_fee) =
                        fees.amount.saturating_mul(Decimal::from(UNIT_BALANCE)).to_u128()
                    {
                        if let Err(_) = Self::transfer_asset(
                            &Self::get_pallet_account(),
                            &beneficiary,
                            converted_fee.saturated_into(),
                            fees.asset,
                        ) {
                            // Push it back inside the internal vector
                            // The above function call will only fail if the beneficiary has
                            // balance below existential deposit requirements
                            if let Err(_) = internal_vector.try_push(fees) {
                                return Err(Error::<T>::UnableToTransferFee)
                            }
                        }
                    } else {
                        // Push it back inside the internal vector
                        if let Err(_) = internal_vector.try_push(fees) {
                            return Err(Error::<T>::FailedToConvertDecimaltoBalance)
                        }
                    }
                }
            }
            Ok(())
        })
        .is_ok(),
        Error::<T>::FeesNotCollectedFully
    );
    Self::deposit_event(Event::FeesClaims { beneficiary, snapshot_id });
    Ok(())
}
```

In this refactor, the `unwrap_or_default` is replaced with `if let Err(_) = ...` pattern. This change improves the robustness of the code by ensuring all error cases are handled and any issues are immediately reported rather than being silenced or ignored.
In addition to the `try_push` operation, the error handling for the `Self::transfer_asset(...)` call was also refined.

Redundant declaration of AllowlistedTokenLimit struct

Duplication of the `AllowlistedTokenLimit` structure in two separate pallets creates potential maintenance issues and future bugs.

ID	PDX-104
Scope	Code Quality
Status	Fixed

Description

The `AllowlistedTokenLimit` structure and its corresponding `get` methods are duplicated in both the `asset-handler` and `pallet-ocex-lmp` pallets, even though they serve the same purpose.

pallets/asset-handler/src/lib.rs:106:

```
pub struct AllowlistedTokenLimit;
impl Get<u32> for AllowlistedTokenLimit {
    fn get() -> u32 {
        50 // TODO: Arbitrary value
```

```
    }
}
```

pallets/ocex/src/lib.rs:126:

```
pub struct AllowlistedTokenLimit;
impl Get<u32> for AllowlistedTokenLimit {
    fn get() -> u32 {
        50 // TODO: Arbitrary value
    }
}
```

Duplicating the get methods for the `AllowlistedTokenLimit` structure in multiple pallets is considered a bad practice, as it can lead to readability and maintainability issues. Moreover, it increases the risk of bugs and vulnerabilities if one `get` method is updated and the other is inadvertently overlooked.

Refer to the related issue PDX-105 for more details on the broader concern of using hardcoded values (magic numbers) throughout the codebase. The redundancy in declaring the `AllowlistedTokenLimit` structure across multiple pallets further exacerbates this issue, increasing the risk of inconsistencies and making it more difficult to maintain the code in the long term.

Recommendation

To enhance maintainability and avoid duplication, it is advised to create a single structure for each logical purpose and use it as a dependency in other pallets.
By consolidating the `AllowlistedTokenLimit` structure into a single location, the codebase will be more readable, maintainable, and less prone to future bugs.

Superfluous Else Clause

In `close_trading_pair` and `open_trading_pair` extrinsics of the `pallet-ocex-lmp` pallet, the redundant empty else clause should be eliminated.

ID	PDX-119
Scope	Code Quality
Status	Acknowledged

Description

Within the `close_trading_pair` and `open_trading_pair` extrinsics of the `pallet-ocex-lmp` pallet, there exist redundant else branches that could be expunged:
pallets/ocex/src/lib.rs:370:

```
 else {
    //scope never executed, already ensured if trading pair exits above
}
```

pallets/ocex/src/lib.rs:402:

```
 else {
    //scope never executed, already ensured if trading pair exits above
}
```

Recommendation

It is recommended to eliminate these superfluous empty else clauses, as they serve no functional purpose.

Superfluous Implementation of Hooks Trait

The default implementation of the `Hooks` trait for the `liquidity` pallet is unnecessary.

ID	PDX-117
Scope	Code Quality
Status	Acknowledged

Description

The code provided below prompts the program to execute default implementations for the methods of the `Hooks` trait:

pallets/liquidity/src/lib.rs:151:

```
#[pallet::hooks]
impl<T: Config> Hooks<BlockNumberFor<T>> for Pallet<T> {}
```

The utilization of the `Hooks` trait without implementing any of its methods does not lead to any immediate detrimental effects. However, it is superfluous and can potentially reduce code readability while increasing complexity.

Recommendation

To improve code clarity and readability, we suggest eliminating the use of the `Hooks` trait in the `liquidity` pallet.
This action will make the code more concise and easier to understand.

Test coverage

The project's test coverage currently stands at **49.82%**, which indicates a noticeable gap in testing. Upon further review, we have discovered that some pallets have been either minimally tested or not tested at all.

The coverage check was carried out on the commit 7d921bfa8e67072bbfd64a393d89ab0468822523.

Test coverage was partially improved after the commit 2d88c867b134ed01606c1b4034b4078b01271239.

ID	PDX-112
Scope	Code Quality / Testing
Status	Fixed

Description

Although certain crates such as `chainbridge` and `support` have been thoroughly tested, others have exhibited a lack of test coverage. Specifically, `router`, `thea`, and `thea-message-handler` pallets have not been subjected to unit testing. It appears that `clients` and `primitives` have also exhibited a lack of test coverage.

We suggest utilizing the `cargo tarpaulin` command to assess code coverage:

```
cargo tarpaulin  --exclude polkadex-client --exclude polkadex-node --exclude node-polkadex-runtime --exclude load-testi
```

The newly generated HTML file contains coverage information for packages:

Covered: 2447 of 4912 (49.82%)

Path	Coverage
orderbook client	526 / 1036 (50.77%)
thea client	76 / 513 (14.81%)
pallets	1442/2717 (53.07%)
primitives	345/714 (48.32%)

When examining the coverage for `pallets`, the following results can be observed:

Covered: 1442 of 2717 (53.07%)

Path	Coverage
asset-handler	240/387 (62.02%)
chainbridge	182/209 (87.08%)
liquidity	58/75 (77.33%)
ocex	407/691 (58.90%)
rewards	122/185 (65.95%)
router	0/123 (0.00%)
support	12/12 (100.00%)
swap	289/523 (55.26%)
thea	0/153 (0.00%)
thea-executor	132/169 (78.11%)
thea-message-handler	0/74 (0.00%)

Recommendation

We recommend addressing the low test coverage in the project's pallets. Implementing a comprehensive test suite, which includes unit testing, is essential for ensuring the security, stability, and maintainability of the project.

We suggest implementing unit tests for all pallets within the project, including the `router`, `thea`, and `thea-message-handler` pallets. Each pallet should possess a comprehensive set of unit tests that cover its functionality, edge cases, and possible error conditions.

Establishing continuous integration (CI) systems to automate the test suite execution is recommended. This practice aids the team in identifying gaps in test coverage, regressions, and areas in need of improvement.

TODO comments in code

This issue involves various `TODO` comments in the codebase that indicate unfinished implementations, potential security concerns, or areas needing further scrutiny.

ID	PDX-114
Scope	Code Quality
Status	Acknowledged

Description

`TODO` comments to be implemented/considered/removed:

- Thea client:
  - Do signature check here
  - What if the local foreign node is not synced yet
  - Send it back to network
  - Get last finalized block hash
  - Get last finalized block hash
  - Get last finalized block hash
  - This is not send
- Orderbook client:
  - handle reputation change
  - Should we respond if we are also syncing???
  - Reduce reputation if else block is happens
  - Fix this in the next release
- Pallets:
  - Need a better error mapping
  - Benchmark on initialize
  - Check if we have 2/3rd authorities signed on this - Fixed
  - Make, <Authorities> indexed by network as key1 and validator setid as key2 - Fixed
  - Make it an offence to not provide network as part of next version
  - what happens if someone changes their session key
  - Clean Storage
  - Check if beneficiary can decode to the correct type based on the given network
  - Super majority check here - Fixed
  - Arbitrary value
  - Arbitrary value
  - Arbitrary value
  - We can do it after release, as an upgrade
  - Check if base and quote assets are enabled for deposits
  - The caller should be of operational council
  - Better documentation
  - Discuss if this is expected behaviour, if not then could this be a potential DDOS?
  - Issue no #2(Reward-Calculation) should modify the map with correct values

- This is not zero
- benchmark this function too and add to proposal weight above
- Benchmark this function too
- assumes we round down to int

- Primitives
  - how to gate this only for testing
- Runtime
  - Check if this is correct?
  - Chnage Polkddex Asset ID
  - Chnage Holder Account

Recommendation

We strongly advise addressing all `TODO` comments within your codebase. These annotations are more than simple reminders; they often contain crucial functionality or improvements that have been deferred for later implementation. In particular, pay close attention to those related to system security such as `Do signature check` and `Supermajority check here`. Neglecting these areas could potentially make your system vulnerable.

Typographical Errors in the Project

This report highlights various typographical errors found throughout the project.

ID	PDX-106
Scope	Code Quality
Status	Fixed

Description

- Correct `PendingWithdrawals` getter name from `get_pending_withdrawls` to `get_pending_withdrawals`: *pallets/asset-handler/src/lib.rs:178*:

```
/// Pending Withdrawals
#[pallet::storage]
#[pallet::getter(fn get_pending_withdrawls)]
pub(super) type PendingWithdrawals<T: Config> = /* ... */
```

Also, fix the usage of this getter in *pallets/asset-handler/src/tests.rs* (3 occurrences).

- Correct the doc string comment `WithdrwalLimitReached` to `WithdrawalLimitReached` *pallets/asset-handler/src/lib.rs:241*:

```
/// WithdrwalLimitReached
WithdrawalLimitReached,
```

- Correct the `WithdrawalFees` getter name from `witdrawal_fees` to `withdrawal_fees`: *pallets/thea-executor/src/lib.rs:97*:

```
/// Withdrawal Fees for each network
#[pallet::storage]
#[pallet::getter(fn witdrawal_fees)]
pub(super) type WithdrawalFees<T: Config> = /* ... */
```

- Correct the doc string comment by changing `parametrise` to `parametrize`: *pallets/thea-executor/src/lib.rs:219*:

```
/// (it's used to parametrise the weight of this extrinsic)
```

Recommendation

We recommend fixing the typographical errors identified and consider using a spell checker extension or tool to catch any further mistakes.

Unmaintained ChainBridge pallet

The ChainBridge Substrate pallet is no longer maintained by its creators, ChainSafe, and the official GitHub repository has been archived.

ID	PDX-107
Scope	Code Quality
Status	Fixed

Description

ChainSafe officially stopped maintaining the ChainBridge pallet on July 12, 2022. The last technical update to the code was made on September 8, 2021, and a final update to the repository was made on July 12, 2022 to notify developers that the library is no longer in use or being maintained. The chainbridge-substrate repository displays the following notice:

> This library is no longer in use and maintenance. All further development related to chainbridge will happen in the new repo. More detailed information about chainbridge-core you can find in its readme or Discussions.
>
> If you already running an old ChainBridge version please consider checking chainbridge-migration scripts that allow migrating to a newer version of chainbridge.

As a result, the Polkadex team is now solely responsible for maintaining and improving this pallet.

Recommendation

We recommend that the Polkadex team consider the following steps to ensure the continued functionality and security of the ChainBridge pallet:

1. Assess the current state of the ChainBridge pallet, identifying any known issues, security vulnerabilities, or areas for improvement.
2. Fork the ChainBridge Substrate repository and assume responsibility for its maintenance and development. This will allow the Polkadex team to address any issues and implement necessary updates.
3. Regularly review and monitor the ChainBridge pallet for potential vulnerabilities or bugs, applying fixes and improvements as needed.
4. If feasible, explore alternatives to the ChainBridge pallet that may provide better support, maintenance, and updates. This could involve evaluating other cross-chain solutions or developing a custom solution tailored to the Polkadex project.

Unnecessary 'hashing.rs' File in Polkadex Chainbridge Pallet

Redundant *hashing.rs* file in the Polkadex `chainbridge` pallet leads to clutter and reduced code readability.

ID	PDX-103
Scope	Code Quality
Status	Fixed

Description

The Polkadex `chainbridge` pallet includes an unnecessary file named *hashing.rs*, which is not utilized in the pallet's implementation. This file contains two public functions, `blake2_128_into` and `blake2_128`, which are not employed in the code. Instead, the Substrate `sp-core` crate provides similar functions that are being used in Polkadex code, making the *hashing.rs* file redundant.

Recommendation

To enhance code readability and minimize clutter, we recommend removing the superfluous *hashing.rs* file from the Polkadex `chainbridge` pallet. This action will contribute to a cleaner codebase, preventing confusion or potential issues that could arise from unused code.

Unsafe arithmetics

Some calculations were performed without considering the possibility of overflows.

ID	PDX-113
Scope	Code Quality / Testing
Status	Acknowledged

Details

In order to ensure proper operation of the nodes, it is necessary to validate all potential errors related to overflows in arithmetic calculations. An overflow in the pallet would result in a node crash, which should be avoided at all costs. Although the code listed below is not expected to cause a crash, it is still recommended to exercise caution in this regard.

Run the following command to view the complete list of unsafe arithmetic operations:

```
cargo clippy -- -W clippy::arithmetic_side_effects
```

Liquidity pallet

pallets/liquidity/src/lib.rs:286, 311, 317:

```
.for_each(|v| result[v.0 + last_index] = v.1);
```

pallets/liquidity/src/lib.rs:287, 319:

```
last_index += 4;
```

Ocex pallet

pallets/ocex/src/lib.rs:981:

```
summary.snapshot_id.eq(&(last_snapshot_serial_number + 1)),
```

Router pallet

pallets/router/src/lib.rs:135:

```
let contains_duplicate = (1..route.len()).any(|i| route[i..].contains(&route[i - 1]));
```

pallets/router/src/lib.rs:261:

```
output_routes.push((route, amounts[amounts.len() - 1]));
```

pallets/router/src/lib.rs:313:

```
amounts[amounts.len() - 1] >= min_amount_out,
```

pallets/router/src/lib.rs:317-318:

```
for i in 0..(route.len() - 1) {
    let next_index = i + 1;
    T::AMM::swap(&trader, (route[i], route[next_index]), amounts[i])?;
}
```

pallets/router/src/lib.rs:326, 385:

```
amounts[amounts.len() - 1],
```

pallets/router/src/lib.rs:376-377:

```
for i in 0..(route.len() - 1) {
    let next_index = i + 1;
    T::AMM::swap(&trader, (route[i], route[next_index]), amounts[i])?;
}
```

Swap pallet

pallets/swap/src/lib.rs:580-583:

```
for i in 0..(path.len() - 1) {
    let (reserve_in, reserve_out) = Self::get_reserves(path[i], path[i + 1])?;
    let amount_out = Self::get_amount_out(amounts_out[i], reserve_in, reserve_out)?;
    amounts_out[i + 1] = amount_out;
}
```

pallets/swap/src/lib.rs:599-603:

```
amounts_in[amount_len - 1] = amount_out;
for i in (1..(path.len())).rev() {
    let (reserve_in, reserve_out) = Self::get_reserves(path[i - 1], path[i])?;
    let amount_in = Self::get_amount_in(amounts_in[i], reserve_in, reserve_out)?;
    amounts_in[i - 1] = amount_in;
}
```

Thea pallet

pallets/thea/src/lib.rs:311:

```
let new_id = Self::validator_set_id() + 1u64;
```

Chainbridge pallet

pallets/chainbridge/src/lib.rs:47:

```
r_id[30 - i] = id[range - 1 - i]; // Ensure left padding for eth compatibility
```

pallets/chainbridge/src/lib.rs:81:

```
else if total >= threshold && self.votes_against.len() as u32 + threshold > total
```

pallets/chainbridge/src/lib.rs:454:

```
let nonce = Self::chains(id).unwrap_or_default() + 1
```

pallets/chainbridge/src/lib.rs:496:

```
RelayerCount::<T>::mutate(|i| *i += 1);
```

pallets/chainbridge/src/lib.rs:506:

```
RelayerCount::<T>::mutate(|i| *i -= 1);
```

pallets/chainbridge/src/lib.rs:525:

```
ProposalVotes { expiry: now + T::ProposalLifetime::get(), ..Default::default() },
```

Clients

clients/orderbook/src/worker.rs:991:

```
if (Utc::now().timestamp() - *when) > 60
```

Recommendation

Our recommendation is to utilize the methods offered by the Rust Standard Library, which provide functionality for safer arithmetic operations, including `checked_add` (`checked_sub`, `checked_mul`, `checked_div`), `saturating_add`, `overflowing_add` and others.

Hacken OÜ
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia
support@hacken.io

Disclaimers

Hacken disclaimer

The code base provided for audit has been analyzed according to the latest industry code quality, software processes and cybersecurity practices at the date of this report, with discovered security vulnerabilities and issues the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functional specifications). The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code (branch/tag/commit hash) submitted to and reviewed, so it may not be relevant to any other branch. 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 contract 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, public bug bounty program and CI/CD process to ensure security and code quality. English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical disclaimer

Protocol Level Systems are deployed and executed on hardware and software underlying platforms and platform dependencies (Operating System, System Libraries, Runtime Virtual Machines, linked libraries, etc.). The platform, programming languages, and other software related to the Protocol Level System may have vulnerabilities that can lead to security issues and exploits. Thus, Consultant cannot guarantee the explicit security of the Protocol system in full execution environment stack (hardware, OS, libraries, etc.)