HACKEN

PENDULUM CHAIN PARTIAL SECURITY ASSESSMENT

Intro

Name	Pendulum Chain
Website	https://pendulumchain.org/
Repository	https://github.com/pendulum-chain/pendulum
Commit	d01528d17b96bf3de72c36deb3800c2ed0cf2afb
Platform	L1
Network	Polkadot / Pendulum
Languages	Rust
Methodology	Blockchain Protocol and Security Analysis Methodology
Lead Auditor	s.akermoun@hacken.io
Main Auditor	n.lipartiia@hacken.io
Approver	l.ciattaglia@hacken.io
Timeline	19.06.2023 - 28.06.2023
Changelog	28.06.2023 (Preliminary Report)
Changelog	14.08.2023 (Draft Final Report)
Changelog	16.08.2023 (Final Report)

Table of contents

Summary

Pendulum is a project that employs the Substrate framework to provide a secure and scalable platform with an emphasis on performance. The scope of this audit primarily focused on the `orml-currencies-allowance-extension` pallet and part of the `faucoco` runtime including the implementation of `ChainExtension`.

Documentation quality

The documentation for the pendulum project is thoroughly executed, offering extensive insight into the `orml-currencies-allowance-extension` pallet and the relevant components of the runtime. A particular highlight is the extensive use of doc strings, which contributes significantly to the overall accessibility and comprehension of the API and code documentation.

While the project provides comprehensive coverage of most elements, one area that could be further detailed is the documentation regarding the sudo pallet. The sudo pallet, due to its potential influence on governance and decentralization, is a crucial aspect of the platform. Although its current documentation is adequate, expanding on the specific functionality, potential impact, and considerations associated with the sudo pallet could further enhance users' understanding. This addition would support informed decision-making and enhance user awareness of potential risks within the project context.

The total Documentation Quality score is 9 out of 10.

Code quality

The pendulum project showcases a solid commitment to high code quality, which is evident in its adherence to Rust and Substrate principles. The code structure demonstrates a clear understanding of Rust idioms and good practices, which makes it easier for contributors to comprehend and participate in the project.

During the audit, the execution of a linting process on the codebase, in conjunction with a manual review, indicated a few minor enhancements that could further boost the code quality. There were some instances of minor linting warnings and deviations from Rust best practices, including the occasional non-idiomatic usage of Rust patterns and a few hardcoded constants. Rectifying these issues would lead to improved maintainability and readability of the code.

In terms of test coverage, the project currently reports a coverage of **22.22%**. Despite this, we noted a lack of testing for the implementation of the chain extension, which plays a critical role in the platform's functioning. Expanding test coverage to encompass this element would lead to a more exhaustive evaluation of its functionality and bolster the system's reliability.

An important consideration is the project's use of an older version of the Substrate framework. While this does not immediately pose a concern, updating to the latest version of Substrate could bring several advantages. These include resolving potential security risks, addressing dependencies, and benefiting from the most recent patches and enhancements.

In sum, the pendulum project maintains a commendable focus on code quality. With minor improvements and a focus on regular updates, it can continue to uphold its high standards and deliver a secure, robust platform.

The total Code Quality score is 8 out of 10.

Architecture quality

The architecture of the pendulum project, which employs the Substrate framework, provides a secure and scalable platform. However, there are areas of concern that require attention to ensure the project's commitment to decentralized governance.

One area of concern is the usage of the sudo pallet within the project. While the inclusion of the sudo pallet during the development and testing phases is understandable, it poses a risk to the decentralization of the system. The sudo pallet grants unilateral power to a single account, potentially compromising the democratic governance mechanism. If the sudo key were to fall into the wrong hands, it could be exploited to manipulate the system. It is therefore crucial to properly document and carefully manage the sudo pallet to ensure transparency and mitigate potential risks. In the long term, it is recommended to consider phasing out the sudo pallet to uphold the project's commitment to decentralized governance.

The architecture quality score is 9 out of 10.

Security score

During the audit, we identified one medium-severity issue and one low-severity issue within the pendulum project, demonstrating the project's strong commitment to security.

The medium-severity issue PDM-006 relates to the chain extension weights. We observed that these weights were not properly defined, which could potentially lead to imbalanced behavior. Addressing this issue is vital to ensure consistent and reliable performance in the chain extension functionality. The issue was fixed in commit 5b922a210b2a6705d3ea6fefbf67b317698f7b80

The low-severity issue PDM-007 concerns the usage of an unconstrained vector. This raises the possibility of excessive memory consumption, potentially opening the door to denial-of-service attacks. By implementing appropriate size restrictions or validation checks, these risks can be mitigated, ensuring the efficient and secure functioning of the system. The issue was partially addressed in the commit c1a20acd965cc024ac756effbff8a12522dac87a, and it was ultimately resolved in the commit 05607a1a9cd2ad3cebeff1294b2e4c34fa3e4721.

Though one issue is of medium severity and the other is low, prompt attention to both will bolster the overall security and reliability of the pendulum project. By doing so, the project continues its commitment to high security standards.

The security score is 10 out of 10.

Total score

Considering all metrics, the total score of the report is **9.6** 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	0	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	1	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	2	

Scope of the audit

Protocol Audit

Chain Extension

- https://github.com/pendulum-chain/pendulum/blob/main/runtime/foucoco/src/lib.rs#L935-L1315

Currencies Allowance Extension pallet

- https://github.com/pendulum-chain/pendulum/tree/main/pallets/orml-currencies-allowance-extension

Issues

`ChainExtension` Implementation Lacks Weight Charging

The current implementation of the `ChainExtension` trait fails to charge weight when allowing smart contracts to call into the runtime.

ID	PDM-006
Scope	Chain Extension
Severity	MEDIUM
Vulnerability Type	Denial-of-Service (DoS)
Status	Fixed (5b922a210b2a6705d3ea6fefbf67b317698f7b80)

Description

The `call` method in the `ChainExtension` trait defines how smart contracts can interact with the runtime. When a contract makes state changes by calling into the runtime, the corresponding weight should be charged. However, the current implementation of `ChainExtension` lacks any weight charging mechanism. It allows smart contracts to make calls such as `transfer`, `approve_transfer`, and `transfer_approved`, which are not queries and result in state changes.

By not charging weight for these operations, the system becomes vulnerable to potential security issues, particularly Denial-of-Service (DoS) attacks. Malicious contracts can exploit this lack of weight charging by flooding the system with a high volume of calls, overloading the network and disrupting its normal operation.

Recommendation

To mitigate this vulnerability, it is crucial to incorporate weight calculations and invoke the `charge_weight` function before accessing contract memory. Here is an example of how to integrate this in the code:

```
let charged_weight = env.charge_weight(sufficient_weight)?;
trace!(
    target: "runtime",
    "[ChainExtension]|call|func_id / charge_weight:{:?}",
    charged_weight
);
let input = env.read(256)?;
```

By implementing weight charging for contract calls, the system can ensure that the appropriate amount of resources is consumed, preventing potential DoS attacks and enhancing overall security.

Vector of unlimited size in the pallet

The `orml-currencies-allowance-extension` pallet employs the usage of the `Vec` data structure without incorporating any size checks.

ID	PDM-007
Scope	`orml-currencies-allowance-extension` pallet
Severity	LOW

Vulnerability Type	Memory exhaustion / DoS
Status	Fixed (c1a20acd965cc024ac756effbff8a12522dac87a and 05607a1a9cd2ad3cebeff1294b2e4c34fa3e4721)

Description

Within the `orml-currencies-allowance-extension` pallet, the `Vec` structure is utilized in the `enum Event` and `struct GenesisConfig`. Additionally, the extrinsic functions `add_allowed_currencies` and `remove_allowed_currencies` accept a vector as a parameter. Here's an example:

pallets/orml-currencies-allowance-extension/src/lib.rs:134:

```
#[pallet::call_index(0)]
#[pallet::weight(<T as Config>::WeightInfo::add_allowed_currencies())]
#[transactional]
pub fn add_allowed_currencies(
    origin: OriginFor<T>,
    currencies: Vec<CurrencyOf<T>>,
) -> DispatchResult {
    ensure_root(origin)?;
    for i in currencies.clone() {
        AllowedCurrencies::<T>::insert(i, ());
    }
    Self::deposit_event(Event::AllowedCurrenciesAdded { currencies });
    Ok(())
}
```

If an excessively large vector is provided as input to these functions, it can result in node overload due to the necessity of iterating through the entire vector. This poses a significant risk, leading to potential memory exhaustion and creating a vulnerability for denial-of-service (DoS) attacks. Although the function is restricted to root access, it's important to note that if the root account is compromised or controlled by a malicious actor, they can exploit this vulnerability by deliberately supplying large vectors. This would cause the node to consume excessive resources, potentially disrupting the normal operation of the system.

Recommendation

To mitigate the risks associated with the use of unlimited-sized vectors, it is strongly recommended to revise the implementation in the `orml-currencies-allowance-extension` pallet.

We advise against the use of the `Vec` data structure within any part of the pallet. Instead, consider utilizing `frame_support::BoundedVec`, which offers a bounded-size variant of a vector. Alternatively, it's important to impose a cap on the length of the vector during each of its instances. By ensuring a maximum limit, the vector's length stays within manageable bounds, preventing excessively long processing times or resource consumption during iterations over this data structure.

By proactively addressing this issue, you can enhance the stability and security of the `orml-currencies-allowance-extension` pallet, ensuring the robustness and reliability of the overall system.

Employment of Sudo Pallet

The current implementation of the sudo FRAME pallet in the runtime employs it as an alternative to the governance mechanism.

ID	PDM-010
Scope	Decentralization
Status	Acknowledged

Description

The `sudo` pallet is integrated into the runtime configuration:

runtime/foucoco/src/lib.rs:1509:

```
construct_runtime!(
    pub enum Runtime where
        Block = Block,
        NodeBlock = opaque::Block,
        UncheckedExtrinsic = UncheckedExtrinsic,
    {
        /* ... */
        // Governance
        Sudo: pallet_sudo::{Pallet, Call, Storage, Config<T>, Event<T>} = 12,
        /* ... */
    }
);
```

The root account, initially set at genesis, is defined as follows: *node/src/chain_spec.rs:229*:

```
let sudo_account =
    pallet_multisig::Pallet::<foucoco_runtime::Runtime>::multi_account_id(&signatories[..], 3);
```

This configuration allows the use of the `ensure_root` method for operations such as `add_allowed_currencies` and `remove_allowed_currencies`. Although there is no immediate security risk associated with this setup, it raises concerns regarding potential centralization.

Recommendation

Utilizing a root account for governance purposes is considered a less favorable design choice due to its potential for centralization and the security risks associated with compromised private keys.

To address this issue, we recommend the following actions:

1. **Comprehensive Documentation:** It is crucial to thoroughly document the intended use of the `sudo` pallet and the root account within the project. This documentation should provide clear explanations of the potential risks and limitations associated with their usage. Both the development team and end-users should be adequately informed to ensure transparency and informed decision-making. If there are plans to deactivate the `sudo` functionality after the network launch, a detailed process similar to the sudo removal outlined by Polkadot should be documented.

2. **Auditing and Monitoring:** Prior to the removal of the `sudo` pallet, it is essential to establish regular auditing and monitoring processes. These measures will help ensure that the `sudo` pallet and the root account are not misused or compromising the system's security. By conducting thorough audits and continuous monitoring, any potential vulnerabilities or misuse can be identified and addressed promptly.

By following these recommendations, you will mitigate potential centralization concerns and enhance the overall governance and security aspects of the project.

Error Handling in Chain Extension

Implementation of `ChainExtension` often utilizes `DispatchError::Other("Explanatory string")` error, which makes error handling difficult.

ID	PDM-009
Scope	Code Quality

Status	Acknowledged

Description

The current implementation of `ChainExtension` frequently relies on the generic `DispatchError::Other()` error. This approach makes error handling challenging, particularly for developers working with smart contracts. The indiscriminate use of this error type makes it difficult to monitor and diagnose specific errors, impeding efficient troubleshooting and code improvement.

Recommendation

To improve error handling in the `ChainExtension` implementation, we recommend implementing a custom enum that covers all the required error types. This custom enum will provide a more structured approach to error handling by encapsulating specific errors and enabling precise identification and resolution of issues.

By implementing the custom enum and incorporating it into the `RetVal` result, developers will have access to more detailed error information, allowing them to effectively identify and handle different error scenarios. This approach enhances the overall robustness and maintainability of the codebase, as well as facilitates troubleshooting and future improvements.

Hardcoded Constants in `match`

The presence of hardcoded constants in the `match` statement within the implementation of `ChainExtension` hampers code readability.

ID	PDM-008
Scope	Code Quality
Status	Acknowledged

Description

The following code segment has drawn our attention:

runtime/foucoco/src/lib.rs:964:

```
match func_id {
    //transfer
    1105 => { /* ... */ },
    //balance
    1106 => { /* ... */ },
    //total_supply
    1107 => { /* ... */ },
    //approve_transfer
    1108 => { /* ... */ },
    //transfer_approved
    1109 => { /* ... */ },
    //allowance
    1110 => { /* ... */ },
    //dia price feed
    7777 => { /* ... */ },
    _ => { /* ... */ },
}
```

While the code comments provide some understanding of the functionality associated with each `func_id`, it is recommended to enhance the code logic in this section to improve comprehensibility, rather than solely relying on comments.

Recommendation

We recommend creating a separate enum to handle all the supported `func_id` options, using descriptive names that correspond to each function. This approach enhances code readability and maintainability:

```
enum FuncId {
    Transfer,
    Balance,
    TotalSupply
    ApproveTransfer,
    TransferApproved,
    Allowance,
    DiaPriceFeed,
}
impl TryFrom<u16> for FuncId {
    type Error = DispatchError;
    fn try_from(func_id: u16) -> Result<Self, Self::Error> {
        let id = match func_id {
            1105 => Self::Transfer,
            1106 => Self::Balance,
            1107 => Self::TotalSupply,
            1108 => Self::ApproveTransfer,
            1109 => Self::TransferApproved,
            1110 => Self::Allowance,
            7777 => Self::DiaPriceFeed,
            _ => {
                error!("Called an unregistered `func_id`: {:}", func_id);
                return Err(DispatchError::Other("Unimplemented func_id"))
            }
        };
        Ok(id)
    }
}
```

This `enum` allows for easier handling of the logic within the `match` statement in the `call` implementation.

Furthermore, it is good practice to implement the logic for each function in a separate method instead of encapsulating all the code within the `match` arms. This promotes code modularity and readability.

Here's an example of how the code would look with the recommended changes:

```
let func_id = FuncId::try_from(env.func_id())?;
match func_id {
    FuncId::Transfer => /* Call method that performs transfer */,
    FuncId::Balance => /* Call method that returns balance */,
    FuncId::TotalSupply => /* ... */ ,
    FuncId::ApproveTransfer => /* ... */,
    FuncId::TransferApproved => /* ... */,
    FuncId::Allowance => /* ... */,
    FuncId::DiaPriceFeed => /* ... */,
}
```

With these improvements, the code becomes more comprehensible, maintainable, and follows best practices.

Linter Warnings

The codebase has generated several warnings when analyzed using `cargo clippy`, indicating areas where improvements can be made to enhance the code quality.

ID	PDM-002
Scope	Linters
Status	Acknowledged

Description

During the static analysis process using `cargo clippy`, the following warnings withing the scope of the audit were identified:

//rust-lang.github.io/rust-clippy/master/index.html#needless_return">needless_return
//rust-lang.github.io/rust-clippy/master/index.html#needless_borrow">needless_borrow

pallets/orml-currencies-allowance-extension/src/lib.rs:245-246:

```
&owner,
&destination,
```

Recommendation

To maintain a high-quality and easily maintainable codebase, it is advised to address the warnings generated by `cargo clippy`. Resolving these linter warnings will improve the overall code quality, facilitate troubleshooting, and potentially enhance performance and security within your the project.

Logging in Runtime

The implementation of `ChainExtension` contains a lot on unnecessary `warn!()` macros.

ID	PDM-011
Scope	Logging
Status	Acknowledged

Description

Upon reviewing the runtime code, it was observed that numerous instances of the `warn!()` macro are present. While the `warn` macro can be useful during testing or debugging phases, it is unnecessary in the final version of the code. These superfluous `warn` macros clutter the codebase and can impede code readability and maintenance.

Unneeded logging statements add noise to the code and distract developers from critical information. In the absence of proper justification, excessive logging can obscure important log messages and make it more challenging to identify and address genuine issues.

Recommendation

To enhance the code quality and maintainability of the runtime, we recommend refining the use of `warn` macros. By limiting the use of these macros to critical and important situations, the codebase will become more concise and easier to comprehend. This targeted approach will enable developers to focus on essential log messages that offer meaningful insights into the system's behavior, rather than being overwhelmed by extraneous warnings. Therefore, unnecessary `warn` macros should be reviewed and removed, with a priority on retaining only those that provide critical and important warnings.

Pendulum build

The Pendulum chain demonstrates a smooth and error-free build process.

ID	PDM-001
Scope	Build Process

Status	Fixed

Description

The build process for the Pendulum chain is efficient and error-free. When executing the `cargo build --release` command, the output confirms a successful build with only one minor compiler warning related to an unused import. The build process adheres to sound Rust coding practices and follows idiomatic conventions.

During the assessment, it was observed that the `orml-currencies-allowance-extension` pallet is not included as a member of the workspace, although it should be.

Recommendation

To address this issue, we recommend adding `pallets/orml-currencies-allowance-extension` to the `members` list in the main `Cargo.toml` file of the workspace.

Please note that this issue does not encompass the results of linting tools, such as `Clippy`, which may provide additional warnings and recommendations for improving code quality. These will be addressed separately in PDM-002.

Superfluous Implementation of Hooks Trait

The `Hooks` trait has been declared in the `orml-currencies-allowance-extension` pallet, but no custom implementations have been provided.

ID	PDM-005
Scope	Code Quality
Status	Acknowledged

Description

The following code snippet raises concerns:

pallets/orml-currencies-allowance-extension/src/lib.rs:99:

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

The `Hooks` trait is typically used to perform logic on every block initialization, finalization, and other specific actions. However, in the case of the `orml-currencies-allowance-extension` pallet, no methods from the `Hooks` trait are implemented. While this does not cause any immediate detrimental effects, it can reduce code readability and increase complexity.

Recommendation

To improve code clarity and readability, it is recommended to eliminate the use of the `Hooks` trait in the `orml-currencies-allowance-extension` pallet. Since no custom implementations are provided, removing the `Hooks` trait declaration will simplify the code and remove unnecessary complexity.

Test Coverage

The current test coverage of the `orml-currencies-allowance-extension` pallet is insufficient, with only **22.22%** coverage. Moreover, the implementation of `ChainExtension` in the runtime lacks any test coverage.

ID	PDM-003
Scope	Code Quality / Testing
Status	Acknowledged

Description

To evaluate the test coverage, we recommend using the `cargo tarpaulin` command:

```
cargo tarpaulin --out Html --output-dir ./tarpaulin-report
```

Running this command generates an HTML file that provides detailed coverage information for all packages. Specifically, it highlights the `orml-currencies-allowance-extension` pallet, which currently has a coverage of only **22.22%**. Additionally, it reveals the absence of tests for the methods utilized in the runtime to implement `ChainExtension`, including `is_allowed_currency`, `allowance`, `do_approve_transfer`, and `do_transfer_approved`, as well as the lack of testing for the overall `ChainExtension` implementation in the `foucoco` runtime.

Recommendation

To address the low test coverage in the `orml-currencies-allowance-extension` pallet and the lack of test coverage in the runtime implementation of `ChainExtension`, it is essential to develop a comprehensive test suite. This suite should include thorough testing to ensure the security, stability, and maintainability of the project.

Implementing continuous integration (CI) systems to automate the execution of the test suite is highly recommended. This practice enables the team to identify areas with insufficient test coverage, detect regressions, and uncover areas that require improvement. By incorporating CI into the development workflow, valuable feedback is obtained, ultimately enhancing the overall quality of the codebase.

Vulnerable and Unmaintained Dependencies

The `orml-currencies-allowance-extension` pallet has dependencies that include one vulnerable crate plus three unmaintained crates.

ID	PDM-004
Scope	Dependencies
Status	Acknowledged

Description

The `orml-currencies-allowance-extension` pallet has several dependencies that raise concerns regarding their security and maintenance. The following table provides details about these dependencies:

Dependency	Version	Id	Type	Remediation
time	**0.1.45**	RUSTSEC-2020-0071	**Vulnerability**	**Upgrade to >=0.2.23**
ansi_term	**0.12.1**	RUSTSEC-	**unmaintained**	**Use alternative crates: anstyle, console, nu-ansi-term,**

		2021-0139		**owo-colors, stylish, yansi**
mach	**0.3.2**	RUSTSEC-2020-0168	**unmaintained**	**Switch to mach2**
parity-wasm	**0.45.0**	RUSTSEC-2022-0061	**unmaintained**	**Switch to wasm-tools**

Although these dependencies may not have an immediate impact on the security aspect and are part of the Substrate project, it is essential to regularly review dependencies and monitor for updates to ensure the overall security and maintenance of the project.

Recommendation

To address these concerns and maintain a secure codebase, it is recommended to take the following actions:

- Use `cargo audit` to check for any new vulnerabilities or outdated packages in your project's dependencies. Regularly perform these checks to stay updated on potential security issues.

- Monitor for new releases of Substrate and update your project accordingly. Keeping your project up to date with the latest Substrate releases ensures that you benefit from bug fixes, security patches, and improvements.

Additionally, it is worth noting that the latest available Substrate release at the time of writing is `v0.9.43`. Stay informed about new releases and evaluate the feasibility of updating your project to benefit from the latest features and improvements.

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.)