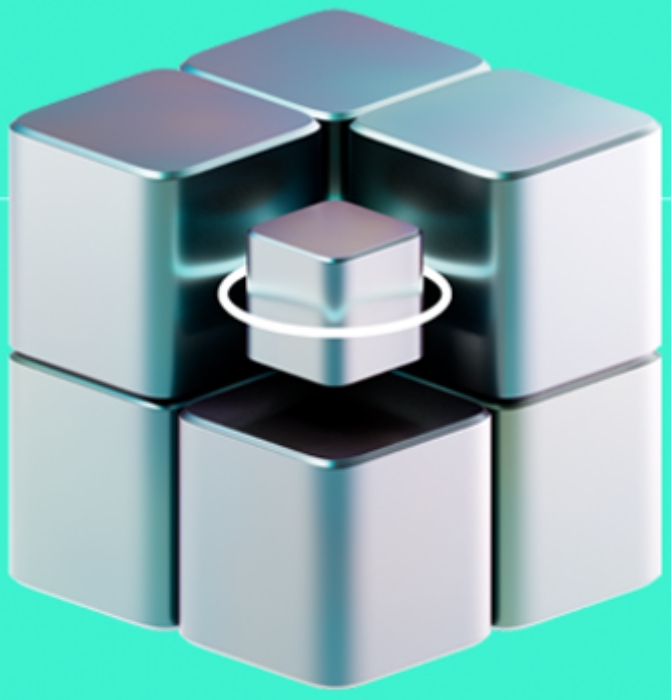
HACKEN

Blockchain Protocol Security Analysis Report



Customer: DENT Wireless Limited

Date: 01/03/2024

We express our gratitude to the DENT Wireless Limited team for the collaborative engagement that enabled the execution of this Security Assessment.

DENTNet represents a revolutionary approach, designed to enhance the management of telecommunication assets through seamless integration with existing telco systems, leveraging blockchain technology.
This innovative solution is tailored for use by mobile operators and their ecosystem of partners, including enterprises, resellers, and service providers.
DENTNet aims to deliver services to users in a manner that is both secure and decentralized, setting a new standard for transparency and efficiency in the telecommunications industry.

Platform: DENTNet

Language: Rust

Tags: Substrate, Bridge

Timeline: 24/01/2024 - 01/03/2024

Methodology: Blockchain Protocol and Security Analysis Methodology

Review Scope

Repository	https://github.com/dentnet/dentnet-node
Commit	24454d3af428f92ebe42341403e0aa551ac6e1d6

Audit Summary

10/10	9/10	10/10	10/10
Security Score	Code quality score	Architecture quality score	Documentation quality score

Total 9.9/10

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

2	2	0	0
Total Findings	Resolved	Accepted	Mitigated

Findings by severity

Critical	0
High	0
Medium	1
Low	1

Vulnerability	Status
F-2024-1062 - Missing exchange rate protection parameter	Fixed
F-2024-1164 - Sponsorees can't remove their sponsorship relationship	Fixed

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.

Document

Name	Blockchain Protocol Code Review and Security Analysis Report for DENT Wireless Limited
Audited By	Nataliia Balashova
Approved By	Sofiane Akermoun
Website	https://www.dentwireless.com/
Changelog	01/03/2024 - Preliminary Report
ChangeLog	08/03/2024 - Final report

Table of Contents

System Overview

DENTNet nodes employ the Substrate framework, including multiple pallets and a specifically configured Runtime. DENTNet nodes utilize the Aura block authoring mechanism and the GRANDPA finality gadget for consensus.

Executive Summary

This report presents an in-depth analysis and scoring of the customer's blockchain protocol project. Detailed scoring criteria can be referenced in the corresponding section of the Blockchain Protocol and Security Analysis Methodology.

Documentation quality

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

- The code adheres to Rust and Substrate documentation standards with complete doc strings.

Code quality

The total Code Quality score is **9** out of **10**.

- Good usage of the Substrate Framework
- Very good code coverage
- The project applies Polkadot linter rules for code quality assurance.
- Two minor mitigated quality issues will be addressed in upcoming releases.

Architecture quality

The total Architecture Quality score is **10** out of **10**.

- Based on the Substrate framework, which enhances security and maintainability
- Good integration of ChainBridge pallet
- Functionalities are well scoped within specific pallets
- No tight coupling in pallets design

Security score

Upon auditing, the code was found to contain **0** critical, **0** high, **1** medium, and **1** low severity issues. The two security issues were quickly resolved, resulting in a perfect security score of **10** out of **10**.

All identified issues are detailed in the “Findings” section of this report.

Summary

The comprehensive audit of the customer's blockchain protocol yields an overall score of **9.9**. This score reflects the combined evaluation of documentation, code quality, architecture quality, and security aspects of the project.

Findings

Vulnerability Details

F-2024-1062 - Missing exchange rate protection parameter - Medium

Description:

The current implementation lacks a protection parameter for the expected output amount in the `exchange` extrinsic of the `pallet-exchange`. As a result, there is a potential vulnerability where the `set_rate` extrinsic can be called and executed between an `exchange` call and its execution, leading to a different amount of output than expected.

The `exchange` extrinsic allows users to swap tokens for native currency based on the current exchange rate. However, without a protection parameter for the expected output amount, the exchange rate can be modified by a concurrent call to the `set_rate` extrinsic, occurring between the initiation and execution of the exchange extrinsic. This can result in the output amount being different from what the user expected, introducing a risk of financial discrepancies.

Users may face unexpected and potentially unfavorable output amounts during token exchanges if the exchange rate is modified by a concurrent `set_rate` call. This could lead to financial losses and negatively impact user trust in the token exchange functionality.

Assets:

- Runtime & Pallets

Status: Fixed

Classification

Severity: Medium

Impact: 3/5

Likelihood: 3/5

Recommendations

Recommendation:

Our recommendation is to introduce a protection parameter in the exchange extrinsic to specify the expected output amount. The suggested name for this parameter would be `expected_output`. This parameter

should be used to validate the output amount against the user's expectations before finalizing the token exchange.

By incorporating the `expected_output` parameter, users gain the ability to predefine the anticipated outcome of the token exchange. This proactive measure not only enhances the transparency of the process but also acts as a preventive measure against potential discrepancies between the expected and actual output amounts.

F-2024-1164 - Sponsorees can't remove their sponsorship relationship - Low

Description:

A flaw was identified in the logic for removing a sponsorship relationship.The current implementation of the `remove_sponsor` function in `pallet-sponsor` restricts the ability to terminate the sponsorship to the sponsor only, neglecting the scenario where the sponsoree (the account being sponsored) wishes to sever this relationship.
This oversight is encapsulated in the conditional check within the match statement:

pallets/sponsor/src/lib:115:

```
match sponsor {
Some(v) if v == caller || v == account => {
Sponsors::<T>::remove(&account);
frame_system::Pallet::<T>::dec_sufficients(&account);
Self::deposit_event(Event::SponsorRemoved { sponsor: caller, account
});
Ok(().into())
},
```

This logic incorrectly uses `v == account` to determine if the sponsor is the account attempting to remove the sponsorship, which is a logical fallacy since `v` represents the sponsor's account ID, and `account` is the sponsoree's account ID.
This misalignment prevents sponsorees from having control over their sponsorship status, potentially leading to undesirable dependencies and misuse of the sponsorship mechanism.

Assets:

- Runtime & Pallets

Status: Fixed

Classification

Severity: Low

Impact: 2/5

Likelihood: 2/5

Recommendations

Recommendation: To rectify this issue and align the functionality with the intended design that allows either party (sponsor or sponsoree) to terminate the sponsorship, the conditional check within the `remove_sponsor` function should be updated.

The corrected condition should verify if the caller is the sponsor or the sponsoree, granting both parties equal authority to dissolve the sponsorship.
The revised logic is proposed as follows:

```
pub fn remove_sponsor(
origin: OriginFor<T>,
account: T::AccountId,
) -> DispatchResultWithPostInfo {
let caller = ensure_signed(origin)?;

let sponsor = Self::sponsor(&account);
match sponsor {
Some(v) if v == caller || caller == account => { // FIXED
Sponsors::<T>::remove(&account);
frame_system::Pallet::<T>::dec_sufficients(&account);
Self::deposit_event(Event::SponsorRemoved { sponsor: caller, account
});
Ok(().into())
},
_ => Err(Error::<T>::NotSponsor.into()),
}
}
```

This adjustment ensures that the function checks whether the caller (the entity executing the transaction) is either the current sponsor of the account (`v == caller`) or the sponsoree itself (`caller == account`), thus correctly implementing the dual-party termination capability.

Observation Details

F-2024-0764 - TODO and FIXME comments in the codebase - Info

Description:

The current issue involves the presence of 2 `TODO` and 5 `FIXME` comments distributed throughout the codebase.
Each serves as a signal for areas requiring meticulous attention, considering potential security implications or specific sections demanding heightened scrutiny for improvement. Addressing these aspects is essential for fortifying the software's robustness and security posture.

This set of comments underscores the dynamic and iterative nature inherent in the ongoing software development process. Identifying and addressing these flagged aspects are crucial steps in fortifying the robustness and security posture of the software project.

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

TODO:

- *pallets/sponsor/src/lib.rs:83* Should this pallet also be marked in system pallet as a consumer of the sponsoring account?
- *runtime/src/lib.rs:2040* this is manual for now someday we might be able to use a macro for this particular key

FIXME:

- *node/src/service.rs:529* #1578 make this available through chainspec
- *pallets/vending/Cargo.toml:28* this is a temporary dependency
- *pallets/vending/src/lib.rs:383* It should be using Preservation::Protect
- *pallets/vending/src/lib.rs:597* this doesn't work because it uses namespace CollectionOwner under the hood
- *pallets/vending/src/lib.rs:624* this doesn't work because it uses namespace CollectionOwner under the hood

Assets:

- Code Quality

Status: Fixed

Recommendations

Recommendation:

It is strongly recommended to give thoughtful attention to all `TODO` and `FIXME` comments within your codebase. These annotations extend beyond simple reminders, often encompassing critical functionalities or deferred improvements. Overlooking these areas could potentially expose vulnerabilities in your system, underscoring the importance of addressing

them promptly to enhance the overall robustness and security of your software.

F-2024-0781 - Test coverage - Info

Description:

Currently, the project faces disparities in test coverage across different pallets.
We've encountered some compilation challenges in the test suites of three essential pallets: Chainbridge, Dentbridge, and Vending.

Addressing these compilation issues is crucial for ensuring a reliable testing infrastructure, contributing to the overall robustness and stability of the identified pallets within our project.

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

Name	Coverage
sponsor	0/26 (0%)
executive-collective	234/337 (69,4%)
exchange	0/50 (0%)

Assets:

- Code Quality

Status: Fixed

Recommendations

Recommendation:

In line with industry best practices, it's advisable to strive for a test coverage target of at least **80%.** This benchmark ensures a comprehensive assessment of pallet functionality and significantly reduces the risk of undiscovered issues. Aimed at enhancing the overall reliability and maintainability of our project, achieving this standard will reinforce our commitment to delivering high-quality, robust software.

F-2024-0948 - Inaccurate weight attribute in add_vendor - Info

Description:

In the provided code for the `add_vendor` extrinsic from `pallet-vending`, an invalid weight attribute has been included within the pallet call. This weight attribute is declared as follows:

pallets/vending/src/lib.rs:454

```
#[pallet::weight(<T as Config>::WeightInfo::update_balance())]
#[pallet::call_index(6)]
pub fn add_vendor(
origin: OriginFor<T>,
vendor: T::AccountId,
) -> DispatchResultWithPostInfo {...}
```

The inclusion of the weight attribute implies a connection between the `add_vendor` extrinsic and the resource consumption associated with the `update_balance` logic.

Upon closer examination, it becomes apparent that the `add_vendor` extrinsic neither directly invokes the `update_balance` logic nor necessitates the associated weight calculation. This inaccuracy in the weight attribute introduces confusion, may misguide developers, auditors, or contributors and has the potential to lead to misinterpretation of the extrinsic's resource consumption.

Assets:

- Weights & Benchmarks

Status: Fixed

Recommendations

Recommendation:

To enhance code clarity and prevent potential misinterpretations, it is advisable to remove the redundant weight attribute from the `add_vendor` extrinsic.

This adjustment ensures that weight attributes accurately represent the true resource consumption associated with the specific logic implemented within the extrinsic.

Furthermore, we suggest including `add_vendor` in the `WeightInfo` trait (*pallets/vending/src/weights.rs:36*)

Additionally, consider adding an appropriate weight attribute, such as `WeightInfo::add_vendor()`, to accurately reflect the resource impact of the `add_vendor` extrinsic.

F-2024-0952 - Missing event for set_allowed_sponsors - Info

Description:

The extrinsic `set_allowed_sponsors` from `pallet-sponsor` plays a crucial role in managing permissions. Triggered by a root (administrator) origin, sets the list of permitted sponsors.

pallets/sponsor/src/lib.rs:128

```
pub fn set_allowed_sponsors(
origin: OriginFor<T>,
allowed_sponsors: BoundedVec<T::AccountId, T::MaxSponsors>,
) -> DispatchResultWithPostInfo {
ensure_root(origin)?;

AllowedSponsors::<T>::put(allowed_sponsors);

Ok(().into())
}
```

However, a significant oversight is present in the code implementation, as it lacks the emission of an associated event when modifying the list of allowed sponsors.

In the provided code implementation, there is no event defined in the `Event` enum for tracking changes to the list of allowed sponsors.

pallets/sponsor/src/lib.rs:151

```
pub enum Event<T: Config> {
/// Sponsor added to account
SponsorAdded { sponsor: T::AccountId, account: T::AccountId },
/// Sponsor removed from account
SponsorRemoved { sponsor: T::AccountId, account: T::AccountId },
}
```

Events serve as a fundamental tool for transparency and observability on the blockchain. They enable external systems, users, and other stakeholders to track and respond to changes in the system's state. In this specific scenario, the absence of an event emission means that there is no clear and public record of modifications to the list of allowed sponsors.

Assets:

- Runtime & Pallets

Status: Fixed

Recommendations

Recommendation:

To address this, it is recommended to introduce an event in the `pallet-sponsor` module's `Event` enum (*pallets/sponsor/src/lib.rs:151)*. The suggested name for this event is `AllowedSponsorsSet.` This name conveys the essence of the event, signifying the establishment or modification of the list of allowed sponsors. Such a naming convention

aligns with best practices and ensures a coherent and understandable representation of the event's purpose in the broader context of the blockchain system.

The emitted event acts as a crucial mechanism for communicating and recording the alteration of the `allowed_sponsors` list. By encapsulating pertinent information about the modification, such as the updated list itself or any other relevant details, the event establishes a comprehensive record of the change.

F-2024-1055 - Linter Warnings - Info

Description:

In the course of the static analysis performed using `cargo clippy`, a significant number of warnings have been generated. The warnings, listed below, serve as valuable signals, point towards potential concerns within the codebase, ranging from maintainability and performance to security. It is important to address all these warnings to establish a high-quality and maintainable codebase.

- needless_borrow
- type_complexity
- useless_conversion
- inconsistent_digit_grouping
- identity_op
- match_like_matches_macro
- collapsible_match
- excessive_precision
- new_without_default
- needless_return
- boxed_local
- field_reassign_with_default
- partialeq_to_none
- manual_retain
- unnecessary_cast
- match_result_ok
- large_enum_variant
- needless_borrows_for_generic_args

It's noteworthy that `cargo clippy` is configured to predominantly issue warnings based on the default set of lints. Nevertheless, there might exist additional concerns that can be brought up by enabling and meticulously examining supplementary lints. These potential issues will be addressed systematically in separate, forthcoming issues.

Assets:

- Code Quality

Status: Fixed

Recommendations

Recommendation:

Ensuring a codebase that is both high-quality and maintainable necessitates the resolution of all warnings issued by `cargo clippy`. Tending to these linter warnings not only improves the overall code quality but also facilitates easier maintenance, troubleshooting, the potential enhancement of performance and security aspects within your project.

To address these warnings effectively, we recommend the following steps:

- Prioritize and Plan:
  Evaluate the impact of each warning on the codebase and prioritize resolution based on severity and significance.
- Review and Refactor:
  Conduct a thorough review of the code sections associated with each warning and implement necessary refactoring, such as eliminating needless borrows, simplifying type complexity, and removing redundant conversions.
- Testing:
  Rigorously test the modified code to ensure that the changes do not introduce new issues and that the overall functionality remains intact.

- Documentation:
  Update relevant documentation to accurately reflect any modifications made to the codebase.
- Continuous Monitoring:
  Establish a routine for regular static analysis using cargo clippy and promptly address any new warnings that arise to sustain optimal code quality.

Taking proactive measures to resolve linter warnings not only results in an enhancement of the overall code quality but also creates a culture of continuous improvement and adherence to Rust's best practices.

F-2024-1151 - Potential Improvements in Documentation - Info

Description: Upon careful examination, the comments and documentation accompanying the project stand out for their clarity and organization. The structured approach lays a solid foundation for navigating and understanding the codebase effectively.
Nonetheless, there is an opportunity for improvement by incorporating more detailed explanations. Specifically, the addition of crate-level documentation and commentary on storage values would be beneficial.

Assets:

- Documentation

Status: Fixed

Recommendations

Recommendation: To maximize the effectiveness of the `cargo doc` tool, it is advisable to expand the documentation to encompass crate-level descriptions and detailed explanations of certain storage values. This enhancement will ensure a more thorough and user-friendly resource for developers, filling in crucial gaps in documentation coverage.

F-2024-1177 - Missing event for add_vendor extrinsic - Info

Description: The extrinsic `add_vendor` in `pallet-vending` doesn't emit an event after storage modification.

Assets:

- Runtime & Pallets

Status: Fixed

Recommendations

Recommendation: To address this, it is recommended to add an event in the `pallet-vending` module's `Event` enum for vendor additions and to emit this event after storage modification in `add_vendor` extrinsic.

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/dentnet/dentnet-node
Commit	24454d3af428f92ebe42341403e0aa551ac6e1d6
Whitepaper	DENT Whitepaper

Components in Scope

Cryptography and Keys

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

XCM

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

Runtime & Pallets

- Runtime implementation review
- Pallets review
- Attack scenarios analysis Weight, race,stack,DoS, state implosion, access control bypass...)

RPC

- RPC implementation review
- Attack scenarios analysis (defaults,DoS, overflows, ..)

Substrate client configuration review

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

Substrate fork review

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

Weights & Benchmarks

- Weight values & benchmarks review

Node Tests

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