Security Audit Report

Zeitgeist Combinatorial Betting and Futarchy Security Audit

v1.0

January 2, 2025

Table of Contents

License

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

THIS AUDIT REPORT WAS PREPARED EXCLUSIVELY FOR AND IN THE INTEREST OF THE CLIENT AND SHALL NOT CONSTRUE ANY LEGAL RELATIONSHIP TOWARDS THIRD PARTIES. IN PARTICULAR, THE AUTHOR AND HIS EMPLOYER UNDERTAKE NO LIABILITY OR RESPONSIBILITY TOWARDS THIRD PARTIES AND PROVIDE NO WARRANTIES REGARDING THE FACTUAL ACCURACY OR COMPLETENESS OF THE AUDIT REPORT.

FOR THE AVOIDANCE OF DOUBT, NOTHING CONTAINED IN THIS AUDIT REPORT SHALL BE CONSTRUED TO IMPOSE ADDITIONAL OBLIGATIONS ON COMPANY, INCLUDING WITHOUT LIMITATION WARRANTIES OR LIABILITIES.

This audit has been performed by

Oak Security GmbH

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security GmbH has been engaged by FORECASTING TECHNOLOGIES LTD. to perform a security audit of Zeitgeist Combinatorial Betting and Futarchy.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following target:

Repository	https://github.com/zeitgeistpm/zeitgeist
Commit	`93f06635570c1959a6c20164213337c97ce010bc`

Scope	The scope was restricted to the following directories:
	`.`
	`└── zrml`
	`  ├── combinatorial-tokens`
	`  └── futarchy`
	
	Additionally, all changes introduced by PR 1364 were included in scope.

Fixes verified at commit	`45175043923cee5f1bdebd93a4502e664f0d8cc4`

	Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

Zeitgeist is a prediction markets platform built on Polkadot that allows users to trade on the outcome of future events. The platform facilitates the creation and trading of prediction market shares, where users can speculate on various outcomes ranging from crypto markets to sports, politics, and other real-world events. It uses a specialized AMM (Automated Market Maker) system designed specifically for prediction markets.

This audit covers Zeitgeist's `combinatorial-tokens` and `zrml-futarchy` pallets, which enable governance decisions through prediction markets. These pallets introduce combinatorial betting pools that allow traders to speculate on multiple interconnected outcomes simultaneously – for example, combining predictions about whether a proposal will be enacted with its expected impact on welfare measures. For each governance proposal, the platform creates paired markets that allow participants to take long or short positions on the outcome depending on whether the proposal is enacted or not. This mechanism follows the "vote on values, bet on beliefs" principle, where traditional voting determines the welfare metrics, while prediction markets determine which policies will best achieve those goals.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending**, **Acknowledged**, **Partially Resolved**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	Medium-High	-
Level of documentation	Medium-High	Detailed documentation is available as whitepapers. Additionally, extensive comments are present explaining most of the critical functions and logic.
Test coverage	Medium-High	It was not possible to calculate test coverage due to errors from the Substrate framework, however, test scenarios were implemented to verify most of the palette code. Note that in `zrml/neo-swaps/src/macros.rs:100`, the precision value of the invariant for `assert_pool_state` was changed from 1 to 2, which may increase the risk of missing important edge cases in the test scenarios.

Summary of Findings

No	Description	Severity	Status
1	Insufficient iteration limit in hash point decompression	Major	Resolved
2	Incorrect logarithm calculation	Major	Resolved
3	Low account costs could facilitate the exhaustion of parachain resources	Major	Resolved
4	Oracle price comparison is prone to manipulation	Minor	Resolved
5	Unverified proposal oracle could facilitate approval of malicious proposal	Minor	Resolved
6	Overflow checks are not enabled for the release profile	Minor	Resolved
7	Faulty test case introduces maintenance risk	Minor	Resolved
8	Unresolved `FIXME` comments in the codebase	Informational	Resolved
9	The project has dependencies on vulnerable crates	Informational	Acknowledged
10	Unresolved `TODO` comments in the codebase	Informational	Resolved
11	Miscellaneous comments	Informational	Resolved

Detailed Findings

1. Insufficient iteration limit in hash point decompression

Severity: Major

In `zrml/combinatorial-tokens/src/types/cryptographic_id_manager/decompressor/mod.rs:60`, hash point decompression is limited to 32 iterations through the `DECOMPRESS_HASH_MAX_ITERS` constant. At the same time, mathematical evidence in line 67 suggests that approximately 507 iterations might be required to find a valid point on the `alt_bn128` curve.

This is problematic because this discrepancy means the `decompress_hash` function may frequently return `None`, even for valid inputs, as it might fail to find a valid `y-coordinate` within the current iteration limit.

This can result in a systemic failure of collection ID generation, effectively causing a denial of service in the combinatorial tokens system.

Recommendation

We recommend increasing the iteration limit to accommodate the mathematically required iterations.

Status: Resolved

2. Incorrect logarithm calculation

Severity: Major

In `zrml/neo-swaps/src/utility.rs:24-34`, the `log_ceil` implementation for `u16` uses `u16::MAX.saturating_sub(1)` to calculate `bits_minus_one`. This is incorrect because it sets `bits_minus_one` to `65534` instead of the intended `15` (16 bits minus one). As a result, the logarithm calculation is wrong for all values, as the `floor_log2` computation uses this faulty value to determine the position of the most significant bit.

We classify this issue as major since it is a fault in a math utility. In addition, this `log_ceil` function is used for the weight computation of the `deploy_combinatorial_pool`, `combo_buy` and `combo_sell` functions causing disproportionate overcharging of the user for the computation costs.

Recommendation

We recommend modifying the `bits_minus_one` calculation to
`let bits_minus_one: u16 = (u16::BITS - 1).saturated_into();`

Status: Resolved

3. Low account costs could facilitate the exhaustion of parachain resources

Severity: Major

In `runtime/zeitgeist/src/parameters.rs:72`, the `ExistentialDeposit`, which is the cost to keep an account active on the parachain, is defined as `5 MILLI ZGT`.

However, since at the time of writing the `ZGT` token price is approximately $0.01, the parachain's storage could be bloated with 100,000 active accounts for only $5 worth of `ZGT` tokens (excluding transaction fees).

Consequently, the parachain's costs of maintaining this data might exceed the initial creation costs, leading to potential DoS scenarios.

Recommendation

We recommend increasing the value of `ExistentialDeposit` to make the exhaustion of parachain resources economically impossible.

For reference, the `ExistentialDeposit` on Polkadot is `1 DOT` which is $6 (at the time of writing).

Status: Resolved

4. Oracle price comparison is prone to manipulation

Severity: Minor

In `zrml/neo-swaps/src/types/decision_market_oracle.rs:49-60`, the `try_evaluate` function determines outcomes by directly comparing spot prices between positive and negative outcomes. This unconstrained spot price comparison approach makes the oracle vulnerable to price manipulation and MEV attacks. There are no safeguards that ensure that the given prices are stable around the evaluation e.g. by calculating a time-weighted average price.

In addition, there are also no absolute or relative constraints on the odds configurable that could be used to further qualify the validity of the outcome. This means that a 10% odd would lose against a 10.001% or a 0.1% odd would win as long as the other is smaller.

Therefore, as an informational addendum to the issue, we highlight that several features could be implemented to impede manipulation or make it more costly, e.g. absolute and relative thresholds. Such thresholds would however render the evaluation as undecided, which might not be intended.

We classify this issue as minor since the creation of decision markets is permissioned, limiting the potential impact of manipulation.

Recommendation

We recommend calculating a time-weighted average price. In addition, we recommend allowing proposal creators to specify absolute and relative thresholds to validate the oracle.

Status: Resolved

5. Unverified proposal oracle could facilitate approval of malicious proposal

Severity: Minor

In `zrml/futarchy/src/pallet_impls.rs:29-52`, the `maybe_schedule_proposal` function checks a proposal's approval by calling `proposal.oracle.evaluate` method.

However, the utilized oracle is part of the `Proposal` structure and is neither verified when submitting the proposal nor when scheduling it. Consequently, a malicious oracle could be specified allowing to arbitrarily approve or reject the given proposal.

We classify this issue as minor since only trusted accounts can currently create proposals. However, this might have major consequences if permissionless proposals are implemented.

Recommendation

We recommend implementing a mechanism that verifies the genuineness of a given oracle, e.g. by adding a whitelist.

Status: Resolved

6. Overflow checks are not enabled for the release profile

Severity: Minor

The crates in scope do not enable the `overflow-checks` flag for the release profile.

Since Rust does not enforce overflow checks by default in the release profile and checked math operations might not always be used throughout the codebase, it is safer to let the execution panic in case of overflow.

Recommendation

We recommend enabling overflow checks in all packages, including those that do not currently perform calculations, to prevent unintended consequences if changes are added in future releases or during refactoring.

Status: Resolved

7. Faulty test case introduces maintenance risk

Severity: Minor

In `zrml/combinatorial-tokens/src/tests/redeem_position.rs:47-49`, the test cases `all_zero` and `all_one` use identical input vectors `[B0, B0, B0]`. This means that despite having different test names suggesting different scenarios, they are testing the same case. This duplication could hide potential issues with the actual `all_one` case that should test a vector of `[B1, B1, B1]`.

This issue is minor since incorrect test cases can lead to maintenance issues and reduced confidence in the test suite's effectiveness, even though they are commonly out of scope.

Recommendation

We recommend amending the `all_one` case to test a vector of `[B1, B1, B1]`.

Status: Resolved

8. Unresolved `FIXME` comments in the codebase

Severity: Informational

The following instances of FIXME comments were identified within the given scope:

- `zrml/combinatorial-tokens/src/lib.rs:304`
- `zrml/neo-swaps/src/lib.rs:1011`

Recommendation

We recommend resolving or removing the given `FIXME` comments.

Status: Resolved

9. The project has dependencies on vulnerable crates

Severity: Informational

Components of the project rely on the following dependencies which were identified to be vulnerable:

- `curve25519-dalek` which is affected by RUSTSEC-2024-0344
- `rustls` which is affected by RUSTSEC-2024-0336

Recommendation

We recommend updating the affected dependencies to the latest version.

Status: Acknowledged

10. Unresolved `TODO` comments in the codebase

Severity: Informational

The following instances of TODO comments were identified within the given scope of this audit (excluding tests):

- `primitives/src/asset.rs:58`
- `zrml/combinatorial-tokens/src/lib.rs:209`
- `zrml/combinatorial-tokens/src/traits/combinatorial_id_manager.rs:32`
- `zrml/futarchy/src/types/proposal.rs:24`
- `zrml/neo-swaps/src/lib.rs:20-21`
- `zrml/neo-swaps/src/lib.rs:1172`
- `zrml/neo-swaps/src/lib.rs:1258`
- `zrml/neo-swaps/src/pool_storage.rs:29`
- `zrml/neo-swaps/src/pool_storage.rs:38`
- `zrml/prediction-markets/src/lib.rs:3066`

Recommendation

We recommend resolving or removing the given `TODO` comments.

Status: Resolved

11. Miscellaneous comments

Severity: Informational

Miscellaneous recommendations can be found below.

Recommendation

The following are some recommendations to improve the overall code quality and readability:

- Magic number `63` in `zrml/futarchy/src/pallet_impls.rs:36` for the scheduling priority of a proposal. We recommend defining a constant for this priority value or adding an appropriate comment explaining its purpose.
- Despite the clear operator precedence in the if-statement in `zrml/neo-swaps/src/math/types/combo_math.rs:275`, we recommend adding parentheses to increase readability.
- We recommend reusing the `r_over_b` variable, defined in `zrml/neo-swaps/src/math/types/math.rs:318`, in the `protected_exp` expression in line `320`.
- In `zrml/neo-swaps/src/lib.rs:1329` and `1459`, there are calls to `unsafe` functions that might affect the reserve. We recommend adding in-line comments that clarify why it is safe to rely on these `unsafe` functions here.
- The same `EXP_NUMERICAL_THRESHOLD` constant is separately defined in two instances in `zrml/neo-swaps/src/math/types/math.rs:38` and `zrml/neo-swaps/src/math/types/combo_math.rs:35`. We recommend defining the constant in one place to avoid potential future ambiguity.

Status: Resolved