Zellic

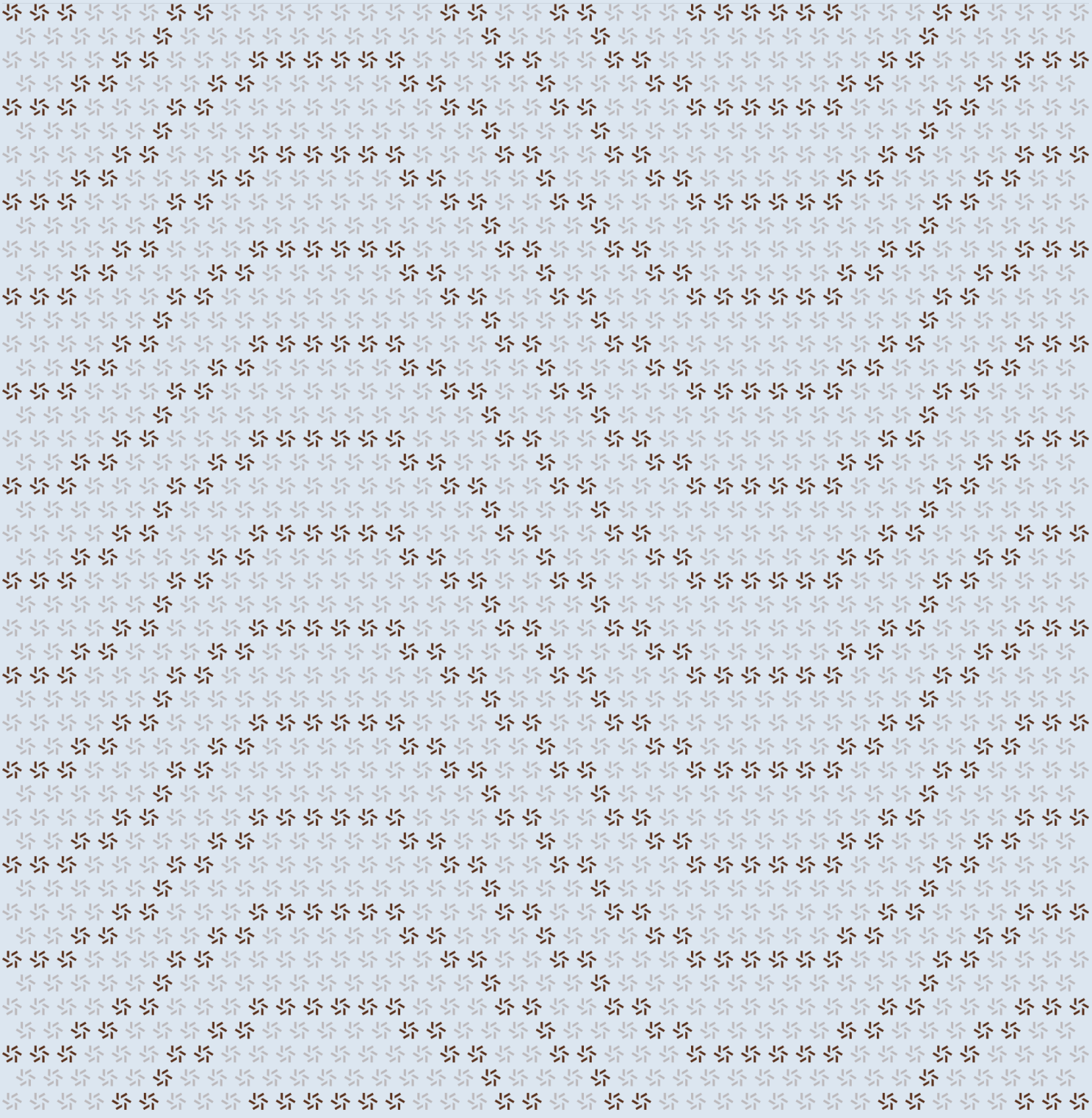
Prepared for
Vince Z
Magma Finance

Prepared by
Sunwoo Hwang
Varun Verma
Zellic

January 31, 2025

Magma Finance

Smart Contract Security Assessment



Contents

About Zellic

Zellic is a vulnerability research firm with deep expertise in blockchain security. We specialize in EVM, Move (Aptos and Sui), and Solana as well as Cairo, NEAR, and Cosmos. We review L1s and L2s, cross-chain protocols, wallets and applied cryptography, zero-knowledge circuits, web applications, and more.

Prior to Zellic, we founded the #1 CTF (competitive hacking) team ↗ worldwide in 2020, 2021, and 2023. Our engineers bring a rich set of skills and backgrounds, including cryptography, web security, mobile security, low-level exploitation, and finance. Our background in traditional information security and competitive hacking has enabled us to consistently discover hidden vulnerabilities and develop novel security research, earning us the reputation as the go-to security firm for teams whose rate of innovation outpaces the existing security landscape.

For more on Zellic's ongoing security research initiatives, check out our website zellic.io ↗ and follow @zellic_io ↗ on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io ↗.

1. Overview

1.1. Executive Summary

Zellic conducted a security assessment for Magma Finance from January 14th to January 27th, 2025. During this engagement, Zellic reviewed Magma Finance's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.2. Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Is there any issue with the CLMM implementation?
- Is there any issue with the access control implementation?
- Is it possible to manipulate the pool price?
- Is it possible to steal other users' liquidity?

1.3. Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Front-end components
- Infrastructure relating to the project
- Key custody

Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide. During this assessment, gaps in response times as well as a lack of developer-oriented documentation and thorough end-to-end testing impacted the momentum of our auditors.

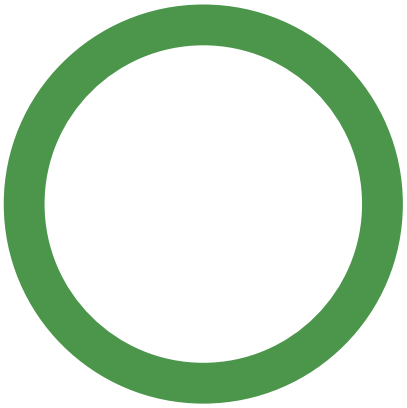
1.4. Results

During our assessment on the scoped Magma Finance contracts, we discovered two findings, both of which were low impact.

Additionally, Zellic recorded its notes and observations from the assessment for the benefit of Magma Finance in the Discussion section (4. ↗).

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	0
Low	2
Informational	0



2. Introduction

2.1. About Magma Finance

Magma Finance contributed the following description of Magma Finance:

> Magma Finance is a cutting-edge AMM DEX designed for MOVE-based blockchains. In Magma, we're combining the classic concentrated liquidity AMM model with the best practice of ve(3,3) tokenomics, providing long-term incentives for early protocol participants and long-term builders.

2.2. Methodology

During a security assessment, Zellic works through standard phases of security auditing, including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

Business logic errors. Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.

Integration risks. Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.

Code maturity. We look for potential improvements in the codebase in general. We look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradability weaknesses, centralization risks, and so on.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case

basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

Finally, Zellic provides a list of miscellaneous observations that do not have security impact or are not directly related to the scoped contracts itself. These observations — found in the Discussion (4. ↗) section of the document — may include suggestions for improving the codebase, or general recommendations, but do not necessarily convey that we suggest a code change.

2.3. Scope

The engagement involved a review of the following targets:

Magma Finance Contracts

Type	Move
Platform	Sui

Target	magma-core
Repository	https://github.com/MagmaFinanceIO/magma-core ↗
Version	8990e48377ba111fa5c7167787cedc00664482e8
Programs	caps/sources/gauge_cap.move clmm/sources/config.move clmm/sources/factory.move clmm/sources/pool.move clmm/sources/position.move clmm/sources/rewarder.move clmm/sources/tick.move distribution/sources/notify_reward_cap.move distribution/sources/reward_authorized_cap.move distribution/sources/reward_distributor_cap.move distribution/sources/team_cap.move distribution/sources/reward_distributor.move distribution/sources/caps/notify_reward_cap.move distribution/sources/caps/reward_authorized_cap.move distribution/sources/caps/reward_distributor_cap.move distribution/sources/caps/team_cap.move distribution/sources/caps/voter_cap.move distribution/sources/common.move distribution/sources/gauge.move distribution/sources/magma_token.move distribution/sources/minter.move distribution/sources/proofs/gauge_to_fee.move distribution/sources/proofs/lock_owner.move distribution/sources/proofs/whitelisted_token_pair.move distribution/sources/rewards/bribe_voting_reward.move distribution/sources/rewards/fee_voting_reward.move distribution/sources/rewards/free_managed_reward.move distribution/sources/rewards/locked_managed_reward.move distribution/sources/rewards/reward.move distribution/sources/voter.move distribution/sources/voting_dao.move distribution/sources/voting_escrow.move

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of 2.8 person-weeks. The assessment was conducted by two consultants over the course of two calendar weeks.

Contact Information

The following project managers were associated with the engagement:

Jacob Goreski
Engagement Manager
jacob@zellic.io

Chad McDonald
Engagement Manager
chad@zellic.io

The following consultants were engaged to conduct the assessment:

Sunwoo Hwang
Engineer
sunwoo@zellic.io

Varun Verma
Engineer
varun@zellic.io

2.5. Project Timeline

January 14, 2025	Start of primary review period
January 27, 2025	End of primary review period

3. Detailed Findings

3.1. Struct not shared

Target	voting_escrow		
Category	Coding Mistakes	**Severity**	Low
Likelihood	Medium	**Impact**	Low

Description

The `Voter` and `VotingEscrow` objects are not properly shared after their creation. This prevents users from accessing these objects when attempting to create a lock and deposit to a gauge, as these operations require access to both the `Voter` and `VotingEscrow` objects.

The issue stems from commented-out code in the `create<T>` function in voting_escrow.move:

```
public fun create<T>(_: &Publisher, voter_id: ID, clock: &Clock, ctx:
    &mut TxContext): VotingEscrow<T> {
    // ... object initialization ...

    // This critical line is commented out:
    // transfer::share_object(voting_escrow);

    voting_escrow
}
```

Impact

Users are prevented from participating in the voting system since they cannot access the required `Voter` and `VotingEscrow` objects for creating locks and depositing to gauges.

Recommendations

We recommend uncommenting the `transfer::share_object(voting_escrow)` line in the create function

Remediation

This issue has been acknowledged by Magma Finance, and fixes were implemented in the following commits:

- 9bbf81d0 ↗
- fa68ff61 ↗

3.2. Wrong check for claimable amount

Target	voter		
Category	Coding Mistakes	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

In the extract_claimable_for function, it checks if the `claimable` amount is greater than the `DURATION`. The `DURATION` is set to `7 * 86400`, which is irrelevant to the claimable amount.

```
fun extract_claimable_for<T>(self: &mut Voter<T>, gauger_id: ID): Balance<T> {
    let gauger_id = into_gauge_id(gauger_id);
    self.update_for_internal<T>(gauger_id); // should set claimable to 0 if
    killed
    let claimable = *self.claimable.borrow(gauger_id);

    // TODO: check if this assert is rigid enough
    assert!(claimable > DURATION);
    // [...]
}
```

Impact

Rewards below the `DURATION` value cannot be claimed, which may prevent reward distribution in certain scenarios.

Recommendations

We recommend removing the check for the claimable amount.

Remediation

This issue has been acknowledged by Magma Finance, and a fix was implemented in commit a4a7a6d8 ↗.

4. Discussion

The purpose of this section is to document miscellaneous observations that we made during the assessment. These discussion notes are not necessarily security related and do not convey that we are suggesting a code change.

4.1. Testing Suite Enhancements for Liquidity Pool Stability

1. Circular Trading Test

The initial test suite lacked validation of complex trading scenarios that simulate real-world activity. To address this, a circular trading test was implemented, executing multiple rounds of back-and-forth trades (10 rounds of A → B followed by B → A swaps). This test verifies that the protocol maintains stable liquidity and reasonable balance ratios under repeated trading pressure.

Key validations:

- Final liquidity matches initial liquidity exactly
- Token balances stay within a 5% deviation

Test Breakdown

1. **Initial Setup**
   - The pool is created with fixed liquidity and starting token balances.
2. **Circular Trading**
   - A series of 10 small round-trip trades is performed within a single price band (tick).
   - Each trade causes minor shifts in token balances, but the price (tick) remains stable since all liquidity is provided in that tick range.
3. **Final State**
   - After all trades, total liquidity remains unchanged.
   - Token balances shift slightly due to accumulated fees and rounding adjustments.
4. **Deviations Analysis**
   - Small increase in Balance A (+3%) and decrease in Balance B (-5%).
   - Deviations fall within acceptable system limits.

---

2. Cross-Tick Trading Test

Building on this, another test was developed to evaluate cross-tick scenarios, focusing on:

- Tick boundary transitions
- Fee accumulation during cross-boundary swaps
- Handling of varying liquidity depths

These tests stress-test the protocol's ability to handle real-world trading patterns while maintaining a consistent state.

Test Breakdown

1. **Pool Initialization**
    - Pool starts with a price at tick 12344.
    - Sufficient liquidity ensures deep markets.
    - Token A and B balances are predefined.
2. **Forward Swaps (Token A → Token B)**
    - Swaps gradually move the tick downward.
    - Each swap (tiny, small, medium, large) shifts the tick step-by-step, consuming liquidity at different price ranges.
    - Larger swaps cause bigger tick movements, as confirmed in debug outputs.
3. **Reverse Swaps (Token B → Token A)**
    - Reverse swaps push the tick back up.
    - The final tick (12199) is close to the initial value, showing minimal drift after a round trip.
4. **Invariants and Deviations**
    - Liquidity remains unchanged.

Observed Core Invariants

- Liquidity is preserved.
- Tick changes remain within acceptable bounds.
- Balance deviations, though high (up to 70%), align with expectations due to fees and rounding effects.

We provided these tests to Magma Finance at the end of the audit.

4.2. Protocol accounting

In the withdraw_managed function within voter_escrow.move, there appears to be an accounting concern around how balances are reduced. The function deducts both the principal amount (weight) and earned rewards (reward_from_locked) from a managed lock's balance in a single calculation: `new_locked_managed.amount - (weight + reward_from_locked)`. This double

deduction could potentially be problematic since the rewards were never included in the initial principal balance.

However, Magma Finance notes this isn't an issue due to how rewards are handled in the implementation. The rewards only originate from the increase_amount_for_internal function (L582 in voting_escrow.move), where for permanent locks, any increased amount is simultaneously added to both the permanent locked balance and treated as managed lock rewards. This design pattern ensures the withdrawal calculation remains accurate despite appearing to double-count the deduction.

4.3. Test suite depth

Is the test coverage adequate?

The current test suite primarily focuses on isolated, single-step operations and basic happy-path flows, such as creating a pool, opening a position, adding liquidity, and removing liquidity. It also verifies that the initial sqrt price falls within the expected range.

However, while these tests cover fundamental functionality, they lack deeper integration and sequential operation testing. More comprehensive coverage of multi-step interactions and complex paths—such as consecutive swaps, liquidity management over time, and protocol-level fee collection—would improve the overall test robustness.

Are edge cases and negative scenarios covered in tests?

There are negative case tests, such as:

- Attempting to remove liquidity without first adding any, which fails as expected.
- Trying to remove zero liquidity, which also fails due to the pool's internal logic—specifically, an attempt to remove from a nonexistent tick raises `ErrTickNotFound`.

Basic edge case checks for liquidity management are present, but additional negative scenario testing across other core functionalities (e.g., swaps, fee distribution, slippage handling) would strengthen coverage.

What assumptions or invariants does the test suite validate?

- A newly created pool has the expected `current_sqrt_price`, `tick_spacing`, and zero liquidity.
- Liquidity removal is only possible if liquidity was previously added.
- A position must contain liquidity; otherwise, removal attempts fail.

Does the test suite simulate real-world usage? (i.e., integration vs. unit testing)

The tests are primarily unit tests, each focusing on a single action or a minimal sequence of steps (e.g., `create pool → open position → add liquidity → remove liquidity`). There is a lack of broader integration tests that simulate real-world interactions over time, such as a sequence of swaps, liquidity shifts, and dynamic fee collection.

Adding tests that cover multiple operations in sequence—such as swapping followed by liquidity adjustments and protocol fee interactions—would provide better insights into how the system behaves under realistic conditions.

Are fuzz tests in place? Would it make sense to add some?

No fuzz tests are present in the codebase. Given the number of parameters that can vary (e.g., swap amounts, tick movements, fee structures, partial liquidity additions), introducing fuzz testing could help uncover edge cases and unexpected behaviors that are difficult to anticipate through traditional test cases.

5. System Design

This provides a description of the high-level components of the system and how they interact, including details like a function's externally controllable inputs and how an attacker could leverage each input to cause harm or which invariants or constraints of the system are critical and must always be upheld.

Not all components in the audit scope may have been modeled. The absence of a component in this section does not necessarily suggest that it is safe.

5.1. Component: CLMM

Description

The core module of the CLMM is composed of `pool.move`, `tick.move` and `position.move`. In pool module, it manages the configuration of pool, including fee and reward rates, and handles all trading operations. Users can swap between two tokens, and open or close positions for token pairs with liquidity to earn trading fees. In tick module, it manages the liquidity of each tick, and calculates each fee for tick range. In position module, it manages the position of each user, and updates each fee and reward.

Invariants

- Users receive an `AddLiquidityReceipt` when they add liquidity to a position. This struct contains the amount of tokens to pay and cannot be dropped, ensuring that the user has to pay the correct amount of tokens.
- The CLMM pool should have correct swap logic for the token pair, ensuring that the liquidity in ticks, amount of tokens, and fees are correct.
- Ticks and liquidity should be updated correctly when the swap amount exceeds the current tick's liquidity (crossing ticks)
- The access control for the pool should be correct, ensuring that only signers with proper capabilities can extract fees from the pool, update fee rates, and pause the pool
- The pool's total liquidity and individual tick liquidity must remain accurate at all times, particularly during tick crossing.
- Each tick is managed by the pool module, ensuring that the tick is not manipulated by other modules.
- Each position is managed by the pool module, ensuring that the position is not manipulated by other modules.

Test Coverage

Cases Covered:

- Open and Close position

- Add and Remove liquidity.
- Stake and Unstake position

Cases not covered

- Swap between two tokens
- Swap crossing the tick
- Collect rewards and fees

5.2. Component: voter.move

Description

The voter.move component manages decentralized governance and reward distribution for liquidity pools. It facilitates voting on liquidity pools using locked assets, distributing rewards based on votes. Ultimately, users can lock assets to gain voting power and allocate votes to pools which influence reward distribution

Invariants

- Each pool has one only gauge
- Weights must sum to 100%
- Rewards only go to alive gauges
- Critical actions require governor/emergency council caps

Test Coverage

Cases Covered:

- The test_vote_before_voting_start and test_vote_after_epoch_voting_end ensure votes fail outside allowed periods
- Checks reward emissions and gauge distribution after epoch advancement
- Tests lock creation and voting power calculation.
- Adding a governor, validating basic governance actions.

Cases not covered

- Potential precision loss in full_math_u64/full_math_u128 not stress-tested
- Killing gauges with active rewards
- No tests for emergency actions

5.3. Component: gauge.move

Description

This component is responsible for:

- Managing gauges, which distribute rewards over time to liquidity positions
- Allowing users to stake and unstake their positions in an associated CLMM pool.
- Tracking each staked position's accrued rewards to claim those rewards.
- Managing the gauge's overall reward rate, period finish, and distribution to staked positions
- Handle fees associated with the gauge in the form of fee_a and fee_b.

Invariants

- A gauge must only work with the pool it was created for
- A position can only be staked in one gauge at a time; cannot be restaked if it is already flagged as staked
- The gauge must be marked as alive in the global config to accept new staked positions
- Only the holder of a valid VoterCap for this gauge can perform admin actions

Test Coverage

Cases Covered:

- Creation of a gauge
- Creating a pool and verifying the gauge's pool_id matches the newly created pool.
- Checking that the position is marked as staked in the pool
- Attempts to deposit a position into a gauge before it is marked alive in config and verify failure occurs

Cases not covered

- Restaking the same position
- Edge cases around timestamps like epoch boundaries

6. Assessment Results

During our assessment on the scoped Magma Finance contracts, we discovered two findings, both of which were low impact.

At the time of our assessment, the reviewed code was not yet deployed to Sui Mainnet. While we have greater confidence in the CLMM component due to more extensive testing, the later scope increase of the voter component resulted in less comprehensive coverage of that aspect. The codebase's complexity and current test coverage suggest additional preparation would be beneficial before mainnet deployment.

Key Areas for Enhancement:

- Code complexity is moderately high, warranting additional validation
- Documentation is somewhat limited, with missing NatSpec and comments that would improve readability
- Test coverage focuses on single actions and could benefit from more multi-step testing to verify correctness of sequential operations, such as the ones we added in 4.1. ↗.

Recommended Next Steps:

- Deploy to Sui Testnet for several weeks to validate intended behavior under real conditions
- Expand test coverage for the voter components by implementing multi-operation test scenarios similar to those we developed for CLMM, rather than relying on single-action tests
- Consider improving code documentation to support long-term maintenance

While the foundation is promising, implementing these recommendations would provide greater assurance for mainnet deployment.

6.1. Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the version reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.