Audit Report

Mercurial AMM

v1.0

05 Octobe 2022

Table of Contents

License

Disclaimer

This audit has been performed by

Oak Security

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

Introduction

Purpose of This Report

Oak Security has been engaged by Dynamic Labs to perform a security audit of their AMM smart contracts for Solana.

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 GitHub repository:

https://github.com/mercurial-finance/mercurial-dynamic-amm

Commit hash: `3a8050396318826ac30cc83d7e2c46bfe4aa074e`

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

The codebade allows the creationf of automated market maker, where all liquidity is stored in the vault to earn extra yield for the liquidity provider. The AMM supports two types of curve, **constant product** and **stable swap** formula.

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**, 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.

Summary of Findings

No	Description	Severity	Status
1	Overwrite amplification parameter is subject to sandwich attack and cause the loss of funds	Critical	Resolved
2	Set pool fees can set arbitrary fee which cause the users to pay a hefty fee	Critical	Resolved
3	Inconsistent behavior on liquidity removal when the pool is disabled	Major	Acknowledged
4	Possible arithmetic operation failure	Major	Resolved
5	Possible arithmetic overflow	Major	Resolved
6	Pool administrative functions can still be invoked while the pool is disabled	Major	Resolved
7	Possible arithmetic overflow	Major	Resolved
8	Failure to check actual account data type	Minor	Acknowledged
9	Confusing match statement causes developer confusion and can lean to potential problems in the future	Minor	Resolved
10	Magic number in space method for Pool	Minor	Acknowledged
11	Wrong Usage of $self in trait implementation causes programming hassle	Informational	Resolved
12	Make space function constant in order to improve performance	Informational	Resolved
13	Wrong Usage of $self in trait implementation causes programming hassle	Informational	Acknowledged
14	Unhandled else condition in stable swap causes confusion	Informational	Acknowledged
15	No test provided for Stable Swap	Informational	Resolved
16	Unhandled else condition in dpedg type causes confusion	Informational	Resolved

17	Wrong Usage of $self in trait implementation causes programming hassle	Informational	Acknowledged
18	Wrong Usage of $self in trait implementation causes programming hassle	Informational	Acknowledged
19	Override Curve parameter does not emit an event	Informational	Resolved
20	TradeDirection enum may be unnecessary	Informational	Acknowledged
21	Enforce/check that the keys are new when calling set_admin_fee_account	Informational	Resolved
22	Move validations to the struct definition that derives the trait Accounts	Informational	Acknowledged
23	Avoid unwrap() and use ? instead	Informational	Resolved
24	Validation can be executed earlier	Informational	Resolved
25	Simplify usage of ok().ok_or() with map_err	Informational	Resolved
26	Alternative (more succinct) implementation of get_amount_by_share that removes unnecessary call to checked_mul	Informational	Resolved

Code Quality Criteria

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	Medium-High	-
Level of documentation	Medium-High	-
Test coverage	Medium-High	-

Detailed Findings

1. Overwrite amplification parameter is subject to sandwich attack and cause the loss of funds

Severity: critical

In mercurial-dynamic-amm/programs/amm/src/lib.rs:889 the `override_curve_param` function will update the amplification factor instantly at a step maximum 10, however, the "maximum 10 change at a time" isn't sufficient to mitigate the sandwich attack. The pool admin can call this functions infinitely many times within a short period, in a matter of seconds or minutes, to move the amplification factor from high to low, which allows the attacker (could be the pool admin himself since anyone can initialize a pool) to execute sandwich trades prior and after the change to gain massive profit by depleting the pool liquidity.

Recommendation

Firstly, consider introducing a damping factor to slowly move amp_old to amp_new in small steps, e.g. increment/decrement 0.1 every hour.

Secondly, consider allowing the function call only after a certain time interval of the previous call, e.g. more than one week after the previous call.

2. Set pool fees can set to an arbitrary fee which causes the users to pay a hefty fee

Severity: critical

In mercurial-dynamic-amm/programs/amm/src/lib.rs:874 the `set_pool_fees` function allows the pool admin to set arbitrary fees, potentially so big that cause the users to pay a high fee right after the fee change.

Recommendation

Firstly, consider setting a global constant that the fee can not go over, e.g. can't be more than 1%.

Secondly, consider adding a damping factor that the new fee will only take effect after a certain time upon invoking the function, e.g. new fee will apply after 7 days calling this function.

3. Inconsistent behavior on liquidity removal when the pool is disabled

Severity: Major

The `remove_balance_liquidity` in mercurial-dynamic-amm/programs/amm/src/lib.rs:663 does not perform a check to validate whether the pool is enabled or not.
However, on the counterparty instruction `remove_liquidity_single_side` through the check in mercurial-dynamic-amm/programs/amm/src/context.rs:169 the instruction prevents the user from removing liquidity if the pool is not enabled.

Recommendation
Enforce the constraint that the pool needs to be enabled on the `remove_balance_liquidity` to prevent users draining the pool from liquidity in case the parameters of the curves or fee aren't correct.

Status: Acknowledged

The team explained that this behavior is justified since the user will be able to remove liquidity from the pool. It is a feature that allows the user to exit their liquidity when an unforeseen event occurs.

4. Possible arithmetic operation failure

Severity: Major

In mercurial-dynamic-amm/programs/amm/src/utils.rs:62~76 the `get_apy` and `u64_to_float` functions use normal deviation operator (`/`). This can break the program in case of denominator equaling 0.
Recommendation

Consider using `checked_div` instead.

Status Resolved
The client handled the case using `safe_float_div` and handling errors, or unwrapping in the caller functions.

5. Possible arithmetic overflow

Severity: Major

In mercurial-dynamic-amm/keeper-amm/src/apy.rs:76~103 the `get_average_apy` function uses mathematical operators such as `+=` and `/` which can result to failure or overflow.
Recommendation

Consider using `checked_div` and `checked_add` instead.

Status Resolved
The client removed the file, and does not use the function any longer.

6. Pool administrative functions can still be invoked while the pool is disabled

Severity: Major

In mercurial-dynamic-amm/programs/amm/src/lib.rs:874 the `set_pool_fees`
In mercurial-dynamic-amm/programs/amm/src/lib.rs:889 the `override_curve_param`
In mercurial-dynamic-amm/programs/amm/src/lib.rs:933 the `transfer_admin`
In mercurial-dynamic-amm/programs/amm/src/lib.rs:943 the `set_admin_fee_account`
In mercurial-dynamic-amm/programs/amm/src/lib.rs:957 the `sync_apy`

These functions allow the pool admin to reset critical pool parameters while pool is disabled, while users can not swap, liquidity providers can not add/remove liquidity. This is a questionable design decision.

The function sync_apy can be invoked while the pool is disabled, this is also questionable.

Recommendation

First, disallow set_pool_fees while the pool is disabled

Second, disable override_curve_param while the pool is disabled.

Third, disable sync_apy while the pool is disabled.

Status Resolved

The client added checks and constraints to ensure that the pool is enabled in all admin related structs.

7. Possible arithmetic overflow

Severity: Major

In mercurial-dynamic-amm/programs/amm/src/curve/curve_stable_swap:53,54 the `pow` function is used instead of `checked_pow`, which can lead to arithmetic overflow issues.
Recommendation

Consider using `checked_pow` instead of `pow`.

Status Resolved

The client replaced `pow` for `checked_pow`.

8. Failure to check actual account data type

Severity: Minor

In mercurial-dynamic-amm/programs/amm/src/dpeg/marinade.rs:31~33 the `get_virtual_price` function deserizes and account data to a `State` .However, it fails to check the actual validity of the data type. This, can lead to unexpected behaviour and program might break in case a malicious users provides the program with a wrong account format.

Recommendation

Consider checking the data type after unwrapping using snipper below.

```
if stake_state.TYPE != Types::State {
            return Err(ProgramError::InvalidAccountType);
}
```

Status: Acknowledged

The team stated that state accounts are validated based on hardcoded addresses. Therefore, there is no possibility to have an invalid account data type.

9. Magic number in space method for Pool

Severity: Minor

In mercurial-dynamic-amm/programs/amm/src/state.rs:173 the `space` function has the **322** magic number which can't be derived from anywhere. It can be that the space is not optimized correctly.

Recommendation

Either introduce a comment to clarify where this number comes from, or a helper function from the elements that account for the space.

Status: Acknowledged

The team included a comment in the code where the number is mapped to the sum of bytes of pool fees.

10. Wrong Usage of $self in trait implementation causes programming hassle

Severity: Informational

In mercurial-dynamic-amm/programs/amm/src/curve/fees.rs:75 the `host_trading_fee` function does not require a PoolFees and does not consume the $self reference in function parameters.
This can be problematic since calling this function would require an instance of PoolFees to be present. Also the function signature does not comply with it's implementation which is in violation of Rust syntax.

Recommendation

Consider removing this parameter so calling the function would not require an instance of `PoolFees` to be passed to the function.

11. Confusing match statement causes developer confusion and can lean to potential problems in the future

Severity: Informational

In mercurial-dynamic-amm/programs/amm/src/lib.rs:184~220 the `swap` function takes advantage of `match` statement to check if `trade_direction` is from A to B or B to A. However, in case of B to A direction instead of explicit value a wilda card `_ =>` has been used. Although considering the possible values for `TradeDirection` enum, this will always evaluate to `TradeDirection::BtoA` , marking this value explicitly can increase code readability. This is problematic for two reasons, first it reduces code readability drastically, and second in case of future updates to `TradeDirection` enum this causes the compiler not to complaint about the code and the logic might end up being incorrect.

Recommendation

Consider removing replacing `_` with `TradeDirection::BtoA`.

12. Make space function constant in order to improve performance

Severity: Informational

In mercurial-dynamic-amm/programs/amm/src/curve/fees.rs:29 the `space` function has not decorated with `const` keywork. Although this is not an issue by any means, taking

advantage of const functions in rust can improve performance, since thise functions are evaluated at compile time instead of run time.

Recommendation

Decorate `space` function with `const` keywork.

***NOTE:** this it sture for all the instances of `space` function throughout the code, since these functions do not use variables and only eachother or other `const` values.

Status Resolved

The client added a `space` attribute using `const`.

13. Wrong Usage of $self in trait implementation causes programming hassle

Severity: Informational

In `mercurial-dynamic-amm/programs/amm/src/curve/curve_stable_swap.rs:304` the `get_default_fee` function does not require a `PoolFees` and does not consume the $self reference in function parameters.
This can be problematic since calling this function would require an instance of `PoolFees` to be present. Also the function signature does not comply with it's implementation which is in violation of Rust syntax.

Recommendation

Consider removing this parameter so calling the function would not require an instance of `PoolFees` to be passed to the function.

Status: Acknowledged

14. Unhandled else condition in stable swap causes confusion

Severity: Informational

In `mercurial-dynamic-amm/programs/amm/src/curve/curve_stable_swap.rs:313~329` the `update_curve_info` function uses an `if let` control flow statement, and returns and `Ok(())` statement at the end. However, in case the if does not match, the control flow lacks an else statement, this can result in unexpected behavior, in case the caller is not aware of this situation.

Recommendation

Consider adding an `else` statement, that returns an error or Zero value.

Status: Acknowledged

The team added a comment on the code explaining that only the depeg stable swap pool needs to update curve information, whereas constant product and stable swap don't

15. No test provided for Stable Swap

Severity: Informational

In `mercurial-dynamic-amm/programs/amm/src/curve/curve_stable_swap.rs` does not contain any test logic, considering the complexity of this module specially in methods such as `swap` which sits at the heart of the system, it is highly recommended to test this module vigorously.

Recommendation

Consider writing tests for stable swap module.

Status Resolved

The team added two tests for the swap module.

16. Unhandled else condition in dpedg type causes confusion

Severity: Informational

In `mercurial-dynamic-amm/programs/amm/src/dpeg/depeg.rs:29~47` the `update_base_virtual_price` function uses an `if let` control flow statement, and returns and `Ok(())` statement at the end. However, in case the if does not match, the control flow lacks an else statement, this can result in unexpected behaviour, in case caller is not aware of this situation.

Recommendation

Consider adding an `else` statement, that returns an error or Zero value.

17. Wrong Usage of $self in trait implementation causes programming hassle

Severity: Informational

In `mercurial-dynamic-amm/programs/amm/src/dpeg/marinade.rs:46~51` the `validate_mint` function does not require a `DepegOperation` and does not consume the $self reference in function parameters.

This can be problematic since calling this function would require an instance of `DepegOperation` to be present. Also the function signature does not comply with it's implementation which is in violation of Rust syntax.

Recommendation

Consider removing this parameter so calling the function would not require an instance of `DepegOperation` to be passed to the function.

18. Wrong Usage of $self in trait implementation causes programming hassle

Severity: Informational

In mercurial-dynamic-amm/programs/amm/src/dpeg/solido.rs:56~61 the `validate_mint` function does not require a `DepegOperation` and does not consume the $self reference in function parameters.

This can be problematic since calling this function would require an instance of `DepegOperation` to be present. Also the function signature does not comply with it's implementation which is in violation of Rust syntax.

Recommendation

Consider removing this parameter so calling the function would not require an instance of `DepegOperation` to be passed to the function.

19. Override Curve parameter does not emit an event

Severity: Informational

In mercurial-dynamic-amm/programs/amm/src/lib.rs:889~930 the `override_curve_param` function does not emit and event. Considering the fact that changing the curve parameter can effects the main calculations, it is recommended to emit an event for this function.

Recommendation

Consider emitting an event.

20. TradeDirection enum may be unnecessary

Severity: informational

In mercurial-dynamic-amm/programs/amm/src/curve/base.rs:48 the `TradeDirection` enum is used widely across the swap and add liquidity and remove liquidity functions in the code, involving the use of verbose "match" statements.

21. Enforce/check that the keys are new when calling set_admin_fee_account

Severity: informational

Following the logic of `transfer_admin` where the instruction validates that the key for the new is not the same than the previous, we believe that the same check can be done for `set_admin_fee_account` in mercurial-dynamic-amm/programs/amm/src/context.rs:421

Recommendation

Instead, we recommend adding these two constraints on `SetAdminFeeAccount` in mercurial-dynamic-amm/programs/amm/src/context.rs:421

```
constraint = pool.admin_token_a_fee != new_admin_token_a_fee.key() @ PoolError::SameAdminAccount
constraint = pool.admin_token_b_fee != new_admin_token_b_fee.key() @ PoolError::SameAdminAccount
```

Status Resolved

The team added a constraint that allows to update either token A, B or both but fails if neither key is new.

22. Move validations to the struct definition that derives the trait Accounts

Severity: informational

There are multiple instances of the code where verifications are performed inside the instruction itself.

mercurial-dynamic-amm/programs/amm/src/lib.rs:540-545
mercurial-dynamic-amm/programs/amm/src/lib.rs:764-769
mercurial-dynamic-amm/programs/amm/src/lib.rs:764-769

Particularly, the check of `AddOfRemoveBalanceLiquidity` is present three times:
mercurial-dynamic-amm/programs/amm/src/lib.rs:770
mercurial-dynamic-amm/programs/amm/src/lib.rs:543
mercurial-dynamic-amm/programs/amm/src/lib.rs:669 (not present, but should!)

Status: Acknowledged

The client confirmed that `AddOrRemoveBalanceLiquidity` struct is shared among `add_balance_liquidity`, `add_imbalance_liquidity`, and `remove_balance_liquidity` instructions. However, when the pool is disabled, there is still a need to allow the user to withdraw liquidity through `remove_balance_liquidity`. Hence, it's best to avoid checking in the struct itself, and keep the validations in the instructions.

Recommendation

Instead, we recommend moving these verifications into the struct definition as a `constraint`.

23. Avoid unwrap() and use ? instead

Severity: informational

There are multiple instances of the code where the error can be safely propagated instead of calling `unwrap()`
mercurial-dynamic-amm/cli-amm/src/main.rs:{836-837, 883, 902}

Recommendation

Replace the unwrap call with ? and handle the error upstream. This is a more idiomatic way of working with errors, especially when the function/method itself returns a `Result<T, E>`

Status Resolved

The team followed the suggestion of replacing the `unwrap` call with ? and handling the errors upstream.

24. Validation can be executed earlier

Severity: informational

mercurial-dynamic-amm/programs/src/lib.rs:819 can be moved to mercurial-dynamic-amm/programs/src/lib.rs:815

Recommendation

This will allow calls to save computation units in case the validation fails.

Status Resolved

The team moved the validations to be performed earlier.

25. Simplify usage of ok().ok_or() with map_err

Severity: informational

mercurial-dynamic-amm/programs/src/lib.rs:1047-1049 chains `ok().ok_or()`

Recommendation

Replace with `map_err()`

Status Resolved

The team followed the suggestion of replacing `ok().ok_or()` with `map_err()`

26. Alternative (more succinct) implementation of get_amount_by_share that removes unnecessary call to checked_mul

Severity: informational

mercurial-dynamic-amm/programs/src/utils.rs:41-58 contains an unnecessary call to `checked_mul`

Recommendation

Our suggestion is to re-use the numerator computation as follows:

```
let num: u128 = share_token.checked_mul(total_amount)?;
let amount: u64 = u64::try_from(num.checked_div(share_token_supply)?).ok()?;
match round_direction {
    RoundDirection::Floor => Some(amount),
    RoundDirection::Ceiling => Some(
        u64::try_from(
            amount
                + u64::try_from(std::cmp::min(v1: 1, v2: num.checked_rem(share_token_supply)?)).ok()?,
        ) Result<u64, Infallible>
        .ok()?,
    ),
}
```

Status Resolved

The team followed the suggestion to re-use the numerator computation.