



SECURITY ASSESSMENT

Provided by Accretion Labs Pte Ltd. for Ellipsis
May 20, 2025
A25ELL1



AUDITORS

Role	Name
Lead Auditor	Robert Reith (robert@accretion.xyz)
Auditor	Niklas Breitfeld (niklas@accretion.xyz)
Auditor	Frank Castle (castlechain99@gmail.com)

CLIENT

Ellipsis (<https://www.ellipsislabs.xyz/>) develops leading DeFi infrastructure and products such as the Phoenix orderbook and the Atlas blockchain. They engaged Accretion to conduct a security assessment of Plasma, a sandwich resistant AMM.

ENGAGEMENT SCOPE

Plasma Program

Link: <https://github.com/Ellipsis-Labs/plasma/>

Commit: 67a8bf4c7c8217d2a9abc7a02cc32aa306edad99

ProgramID: srAMMzfVHVAtgSJc8iH6CfKzuWuUTzLHVCE81QU1rgi

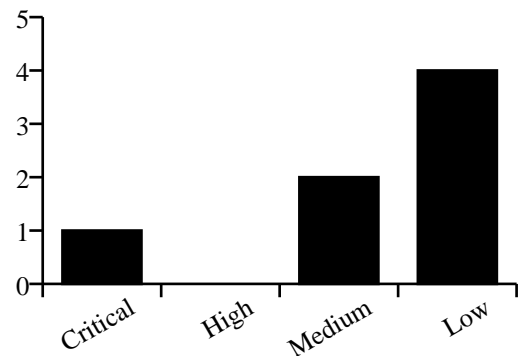
ENGAGEMENT TIMELINE

22 Apr	Project Kickoff Initial planning and scope definition
22 Apr	Assessment Begins Security review and testing phase
01 May	Review Fixes Security recommendations are given and implemented
20 May	Project Completion Report delivery

ASSESSMENT

The security assessment of Plasma revealed a few vulnerabilities primarily located within the liquidity transfer operation, which was the latest and only previously-unaudited addition to the program. The overall program is well engineered and uses a custom framework to validate accounts and inputs. The program itself implements a thoughtful design to prevent sandwich attacks. It also employs a unique design where the AMM collects LP and protocol fees in the quote token only. All issues were quickly and competently fixed by the Ellipsis team.

SEVERITY DISTRIBUTION



AUDITED CODE

Program 1

ProgramID: srAMMzfVHVAtgSJc8iH6CfKzuWuUTzLHVCE81QU1rgi

Repository: <https://github.com/Ellipsis-Labs/plasma/>

Build Hash:

1da2f79548ea5c8ba50c61307385304476ff231ec9e42f4f1124b79578aeb41

Commit: b0600733006c90fc21b3c822c94c8d077676ebb0

ISSUES SUMMARY

ID	TITLE	SEVERITY	STATUS
ACC-C1	Loss of Funds in Liquidity Transfer Mechanism	critical	fixed
ACC-M1	When Selling with Exact out, A Larger Amount Can be Swapped Through The Limit Order than It Holds	medium	fixed
ACC-M2	Incorrect calculation of vesting time passage	medium	fixed
ACC-L1	Broken invariant when force vesting shares	low	fixed
ACC-L2	Fully re-vested shares become unvested again when transferring LP positions	low	accepted
ACC-L3	Unauthorized Liquidity Transfer Can Cause DoS for Liquidity Addition	low	fixed
ACC-L4	Value truncation on pool creation leads to incorrect event reporting	low	fixed
ACC-I1	AMM operations are Asymmetric	info	accepted

DETAILED ISSUES

ID	ACC-C1
Title	Loss of Funds in Liquidity Transfer Mechanism
Severity	critical
Status	fixed

Description

The `transfer_liquidity` function in ``crates/plasma_state/src/lp.rs`` doesn't preprocess the destination LP position before the transfer, leading to loss of funds and reward factor manipulation.

The fee collection mechanism works like this:

- each LP position has 3 relevant variables.
- LP shares which is how many shares it owns
- Reward Factor Snapshot which is the last used Reward Factor
- uncollected fees, which is the amount of fees we can withdraw

Whenever we call `preprocess_lp_position`, the uncollected fees is updated with all the fees that have been accumulated in the time since the last call to this function. For this, we take the current reward factor from the AMM, and subtract the reward factor snapshot in the LP position from it. The difference represents the newly accumulated fees per share for that pool since the last update. We then calculate our uncollected fees by multiplying the reward factor difference with the number of LP shares of the position. The reward factor snapshot is updated to the latest reward factor from the AMM.

This function has to be called whenever new `lp_shares` are added to a position, otherwise we can inflate our withdrawable fees by initializing a position with no shares, wait a long time until the reward factor diff is high, and then add a lot of shares to the position and withdraw fees as if these shares were locked in the position for the whole time.

However, the `transfer_liquidity` function does not call ``preprocess_lp_position`` on the destination LP position, which means we can add LP shares to a position in exactly this malicious way.

An attacker can create multiple empty LP positions, wait for an extended time period, and then create a very large LP position. From this, they withdraw fees, transfer to one of their old empty LP positions, withdraw fees from this, transfer to the next one, withdraw fees, and so on, fully draining a pool.

Location

https://github.com/Ellipsis-Labs/plasma/blob/b371a6cebece41d9fac476e22392f3ab381750e7/crates/plasma_state/src/lp.rs#L176-L183

Relevant Code

```
/// plasma_state/src/lp.rs L176-L183
pub fn transfer_liquidity(
    &mut self,
    slot: SlotWindow,
    amm: &Amm,
    dst: &mut LpPosition,
) -> Result<u64, PlasmaStateError> {
    self.preprocess_lp_position(slot, amm)?;
```

Mitigation Suggestion

Add `dst.preprocess_lp_position(slot, amm)?;` before adding shares to the destination position, to also preprocess and accumulate fees for it.

Remediation

Fixed in commit `92d02bdd3c0063ec653dafa347ff6736fd26fc04`.

ID	ACC-M1
Title	When Selling with Exact out, A Larger Amount Can be Swapped Through The Limit Order than It Holds
Severity	medium
Status	fixed

Description

We found that in the `swap_exact_out` version of swapping, the check if the swap has to be done only using the Bid does a comparison between the size of the bid and the `quote_out` amount, but then proceeds to do the swap through the bid using the `quote_out_pre_fee` amount, which is higher. This means in the case where `size_on_bid_in_quote == quote_out`, we will do a swap through the bid with a larger size than what the bid is for. This practically means we can cross the full spread when selling, and sell the fee at the limit order price as well.

Location

https://github.com/Ellipsis-Labs/plasma/blob/b371a6cebece41d9fac476e22392f3ab381750e7/crates/plasma_state/src/amm.rs#L907-L913

Relevant Code

```
/// amm.rs L907-L913
) = if size_on_bid_in_quote >= quote_out {
    let quote_swapped_through_bid = quote_out_pre_fee;
    let base_swapped_through_bid = self.get_complementary_limit_order_size(
        quote_swapped_through_bid,
        Side::Sell,
        TokenType::Quote,
    );
```

Mitigation Suggestion

The check should be `) = if size_on_bid_in_quote >= quote_out_pre_fee {`.

Remediation

Fixed in commit `119a1744c5f05cd08ed734681334fea2f2576011`.

ID	ACC-M2
Title	Incorrect calculation of vesting time passage
Severity	medium
Status	fixed

Description

We found that when the function `maybe_vest_shares` is called, it incorrectly calculates whether the current slot is eligible for vesting. In particular, the case `if self.deposit_slot + amm.lp_vesting_window <= slot {` is intended to describe the scenario where shares have been vested, and the vesting has completed. However, it confuses different types. `self.deposit_slot` is assigned in ``set``, and is given the value ``slot``, which is a ``SlotWindow``, which is always calculated to be the first slot of the current leader's current 4 slots. It is calculated as follows:

```
/// liquidity.rs L135-L135
let slot = (Clock::get()?.slot / LEADER_SLOT_WINDOW) * LEADER_SLOT_WINDOW;
```

`amm.lp_vesting_window` however is calculated by dividing ``vesting_slot_window`` by 4. It is the third parameter for ``Amm::new`` here:

```
/// initialize.rs L188-L195
pool.amm = Amm::new(
    lp_fee_in_bps as u32,
    protocol_fee_allocation_in_pct as u32,
    vesting_slot_window
        .map(|v| v / LEADER_SLOT_WINDOW)
        .unwrap_or(2),
    slot,
);
```

Lastly, `slot` also describes a ``SlotWindow`` starting slot like ``self.deposit_slot``. This means that when a user creates a Pool with a 4000 vesting slot window, which means that they intended for 4000 slots to pass before LP shares are vested, `maybe_vest_shares` will already allow shares to vest after 1000 slots instead.

Location

https://github.com/Ellipsis-Labs/plasma/blob/b371a6cebece41d9fac476e22392f3ab381750e7/crates/plasma_state/src/lp.rs#L36-L48

Relevant Code

```
/// plasma_state/src/lp.rs L36-L48
pub fn maybe_vest_shares(&mut self, slot: SlotWindow, amm: &Amm) -> u64 {
    if self.deposit_slot == 0 {
        return 0;
    }
    if self.deposit_slot + amm.lp_vesting_window <= slot {
        let lp_shares = self.lp_shares_to_vest;
        self.deposit_slot = 0;
        self.lp_shares_to_vest = 0;
        lp_shares
    } else {
        0
    }
}
```

Mitigation Suggestion

Instead of `if self.deposit_slot + amm.lp_vesting_window <= slot {` use ``if self.deposit_slot + amm.lp_vesting_window * LEADER_SLOT_WINDOW <= slot {``.

Remediation

Fixed in commit `69a2e719e8ce3e60fa8fe7e2b7213be210e3b222`.

ID	ACC-L1
Title	Broken invariant when force vesting shares
Severity	low
Status	fixed

Description

We found that `force_vest_shares` does reset the ``deposit_slot`` to 0, but does not reset ``lp_shares_to_vest`` to 0. Just before calling this function, an invariant is checked:

```
// lp.rs L190-L198
let total_withdrawable_lp_shares =
    self.withdrawable_lp_shares + self.pending_shares_to_vest.lp_shares_to_vest;

if total_withdrawable_lp_shares != self.lp_shares {
    return Err(PlasmaStateError::InvariantViolation(
        total_withdrawable_lp_shares as u128,
        self.lp_shares as u128,
    ));
}
```

This invariant becomes invalid after the function call, as the following two values are set to 0, but `lp_shares_to_vest` is not:

```
// Zero out the source liquidity position
self.withdrawable_lp_shares = 0;
self.lp_shares = 0;
```

As it is an invariant, it should not become invalid after the instruction ends. Furthermore, the `is_vesting` function will stay true for the source LP, even though all shares have been withdrawn.

Location

https://github.com/Ellipsis-Labs/plasma/blob/b371a6cebece41d9fac476e22392f3ab381750e7/crates/plasma_state/src/lp.rs#L51-L55 https://github.com/Ellipsis-Labs/plasma/blob/b371a6cebece41d9fac476e22392f3ab381750e7/crates/plasma_state/src/lp.rs#L22-L24

Relevant Code

```
/// lp.rs L51-L55
pub(crate) fn force_vest_shares(&mut self) -> u64 {
    let lp_shares = self.lp_shares_to_vest;
    self.deposit_slot = 0;
    lp_shares
}

/// lp.rs L22-L24
pub fn is_vesting(&self) -> bool {
    self.lp_shares_to_vest > 0 || self.deposit_slot != 0
}
```

Mitigation Suggestion

Set `lp_shares_to_vest` to 0 when force vesting shares.

Remediation

Fixed in commit `dfc3e354afc51788904c93981503d7665e1cd219`.

ID	ACC-L2
----	--------

Title	Fully re-vested shares become unvested again when transferring LP positions
-------	---

Severity	low
----------	-----

Status	accepted
--------	----------

Description

When a LP position is transferred, the entire source LP balance (lp_shares_transferred) is re-vested at the destination, instead of just the shares that weren't vested yet at the source.

This means if part of the source LP shares were already withdrawable (i.e., not locked under vesting), those shares should not be subjected to a new vesting period upon transfer. Only the pending, vesting LP shares should be re-vested.

Location

https://github.com/Ellipsis-Labs/plasma/blob/b371a6cebece41d9fac476e22392f3ab381750e7/crates/plasma_state/src/lp.rs#L201-L212

Relevant Code

```
/// lp.rs L201-L212
    self.pending_shares_to_vest.force_vest_shares();

    let lp_shares_transferred = self.lp_shares;

    // Zero out the source liquidity position
    self.withdrawable_lp_shares = 0;
    self.lp_shares = 0;

    // Increment the destination liquidity position
    dst.pending_shares_to_vest
        .set(slot, lp_shares_transferred)?;
    dst.lp_shares += lp_shares_transferred;
```

Mitigation Suggestion

Use only the currently vesting amount from the source position during the transfer and add the withdrawable LP shares directly to the destination's LP shares.

Remediation

This issue has been accepted as desired behavior.

ID	ACC-L3
Title	Unauthorized Liquidity Transfer Can Cause DoS for Liquidity Addition
Severity	low
Status	fixed

Description

Any user can transfer a liquidity position to another user's position without permission from the destination owner, leading to a potential **denial-of-service** for that position.

The root cause is that the destination position's owner **does not sign** the liquidity transfer transaction. As a result, liquidity can be forcefully transferred into any position without approval, and the affected position will be **locked** from adding new liquidity until the vesting period ends.

An attacker can continuously transfer the minimum amount of liquidity — at virtually no cost — to a user's position at the end of each vesting window, **perpetually blocking** any actions that depend on the vesting period ending.

Furthermore, the actual minimum amount of liquidity to transfer is 0. In practice, a user can create an LP position with 0 `lp_shares`, and use it to transfer and lock another user's LP position continuously.

The practicality of this attack depends on the pools `lp` vesting window. When this window is set particularly high, this reduces the attacker's transaction costs to permanently DoS a position. If the window however is low as the default of 2 suggests, performing this DoS becomes less feasible for an attacker.

Location

https://github.com/Ellipsis-Labs/plasma/blob/b371a6cebece41d9fac476e22392f3ab381750e7/crates/plasma_state/src/lp.rs#L26-L34

Relevant Code

```
/// lp.rs L26-L34
pub fn set(&mut self, slot: SlotWindow, lp_shares: u64) -> Result<(), PlasmaStateError> {
    if self.deposit_slot == 0 {
        self.deposit_slot = slot;
        self.lp_shares_to_vest = lp_shares;
        Ok(())
    } else {
        Err(PlasmaStateError::VestingPeriodNotOver)
    }
}
```

Mitigation Suggestion

We suggest enforcing the recipients approval for liquidity transfers. This could be done either by requiring the recipients signature on the transfer transaction, or by adding a flag within LP positions which enables or disables incoming liquidity transfers, so that users who are not expecting such a transfer can prevent the position from being locked up.

Remediation

Fixed in commit 69a2e719e8ce3e60fa8fe7e2b7213be210e3b222 by only allowing liquidity transfers to empty LP positions.

ID	ACC-L4
Title	Value truncation on pool creation leads to incorrect event reporting
Severity	low
Status	fixed

Description

When initializing a pool, `protocol_fee_allocation_in_pct` is truncated from a `u64` to a `u32` in the `Amm`, but not elsewhere. For example the created Event does not truncate the value, which results in an Event reporting a different value than what will be stored on chain. The best fix would be to make the value a `u32` from the beginning.

Location

<https://github.com/Ellipsis-Labs/plasma/blob/b371a6cebece41d9fac476e22392f3ab381750e7/program/src/program/processor/initialize.rs#L188-L195>

Relevant Code

```
/// initialize.rs L188-L195
pool.amm = Amm::new(
    lp_fee_in_bps as u32,
    protocol_fee_allocation_in_pct as u32,
    vesting_slot_window
        .map(|v| v / LEADER_SLOT_WINDOW)
        .unwrap_or(2),
    slot,
);
```

Mitigation Suggestion

We recommend changing the type within `InitializePoolParams` from ``u64`` to ``u32`` so that the same type is used throughout the function without any truncation.

Remediation

Fixed in commit `69a2e719e8ce3e60fa8fe7e2b7213be210e3b222` by introducing a bounds check.

ID	ACC-I1
Title	AMM operations are Asymmetric
Severity	info
Status	accepted

Description

As the protocol always charges fees in the Quote token, this automatically leads to asymmetric operations. When the user deposits quote tokens, they get charged immediately on the input. This means (Input-Fee) gets swapped. When the user deposits base tokens, they get charged on the output. This means the full Input is swapped, and then the Fee is deducted on the Output.

This design results in an asymmetric AMM.

Example setup: X : base token , Y: quote token

```
Token X and Token Y
X reserve = 500,000
Y reserve = 500,000
K = 500,000 * 500,000 = 250,000,000,000
```

****Buy operation on Pool A**:** (Quote in) • X bought: 100,000 • Y sold: 125,000 • LP fee: 375 • Protocol fee: 125 • New reserves: X = 400,000, Y = 625,375 • New K: 400,000 * 625,375 = 250,150,000,000

****Sell operation on Pool B**:** (Quote out) • X sold: 125,000 • Y bought: 100,000 • LP fee: 300 • Protocol fee: 100 • New reserves: X = 625,000, Y = 400,300 • New K: 625,000 * 400,300 = 250,187,500,000

Observation: • K value increases differently for each operation. • We charge different fees depending on the side of the operation.

Location

https://github.com/Ellipsis-Labs/plasma/blob/b371a6cebece41d9fac476e22392f3ab381750e7/crates/plasma_state/src/amm.rs#L516-L517

Relevant Code

```
/// amm.rs L516-L517
let quote_fee = self.fee_rounded_down(quote_in.upcast());
let quote_in_post_fee: u128 = quote_in.upcast() - quote_fee;
```

Mitigation Suggestion

If this behavior is not intentional, we recommend making AMM operations symmetrical. This can be done in multiple ways, for example by always charging fees in the input token instead of the quote token.

Remediation

This issue has been accepted as the intended behavior.

APPENDIX

Vulnerability Classification

We rate our issues according to the following scale. Informational issues are reported informally to the developers and are not included in this report.

Severity	Description
Critical	Vulnerabilities that can be easily exploited and result in loss of user funds, or directly violate the protocol's integrity. Immediate action is required.
High	Vulnerabilities that can lead to loss of user funds under non-trivial preconditions, loss of fees, or permanent denial of service that requires a program upgrade. These issues require attention and should be resolved in the short term.
Medium	Vulnerabilities that may be more difficult to exploit but could still lead to some compromise of the system's functionality. For example, partial denial of service attacks, or such attacks that do not require a program upgrade to resolve, but may require manual intervention. These issues should be addressed as part of the normal development cycle.
Low	Vulnerabilities that have a minimal impact on the system's operations and can be fixed over time. These issues may include inconsistencies in state, or require such high capital investments that they are not exploitable profitably.
Informational	Findings that do not pose an immediate risk but could affect the system's efficiency, maintainability, or best practices.

Audit Methodology

Accretion is a boutique security auditor specializing in Solana's ecosystem. We employ a customized approach for each client, strategically allocating our resources to maximize code review effectiveness. Our auditors dedicate substantial time to developing a comprehensive understanding of each program under review, examining design decisions, expected and edge-case behaviors, invariants, optimizations, and data structures, while meticulously verifying mathematical correctness—all within the context of the developers' intentions.

Our audit scope extends beyond on-chain components to include associated infrastructure, such as user interfaces and supporting systems. Every audit encompasses both a holistic protocol design review and detailed line-by-line code analysis.

During our assessment, we focus on identifying:

- Solana-specific vulnerabilities
- Access control issues
- Arithmetic errors and precision loss
- Race conditions and MEV opportunities
- Logic errors and edge cases
- Performance optimization opportunities
- Invariant violations
- Account confusion vulnerabilities
- Authority check omissions
- Token22 implementation risks and SPL-related pitfalls
- Deviations from best practices

Our approach transcends conventional vulnerability classifications. We continuously conduct ecosystem-wide security research to identify and mitigate emerging threat vectors, ensuring our audits remain at the forefront of Solana security practices.