Security Audit - Jupiter Lend DEX

conducted by Neodyme AG

Lead Auditor:	Benjamin Walny
Second Auditor & Administrative Lead:	Jasper Slusallek

June 25, 2026



1 | Operational Security Review

As part of a larger audit engagement, Neodyme reviewed Jupiter Lend's operational security (OpSec) as it pertains to the DEX contract. This report outlines the OpSec parts of this engagement.

The scope of this OpSec review was:

1. the categorization and risk classification of the different program authorities,
2. developing a potential suggestion for a more fine-grained authority split, and
3. a quick review of the team's current multisig and private key hygiene, their resilience against social engineering and their monitoring setup, as well as other key OpSec factors.

Number 1 was necessary to understand the risk that a compromise of each of the authorities represents. Number 2 was done to enable better separation of risk. Obviously, high-risk operations (e.g. operations that allow the operator to withdraw money) should be separated from operations that are done with less care (e.g. setting a non-vital parameter). Number 3 captured the general operational security stance.

We detail part of the results below. Due to operational security concerns, we do not publish all details. We begin with an overview of the contract's authorities:

Contract Authorities

The DEX program has the following relevant authorities:

- **Upgrade Authority:** As with any Solana contract, this authority can change the on-chain bytecode. If an attacker gets control of it, they can thus obviously steal all funds. Right now, the plan appears to be to point this to the global upgrade authority (`4MsgBB5VPoTrUSp5XnfbViV386C1UnsTdifLBw33ZMSJ`), which is a 4-of-7 multisig run by Jupiter and Fluid team members with a 12-hour time delay.
- **Program Init Authority:** According to documentation, this authority is meant to be temporary and is used to create new DEX trading pools or set up a DEX position for protocols. It is held by the Fluid team. While it is copied across other lend protocols and hasn't been removed yet, its power is very limited. If compromised, an attacker could frontrun pool creation or lock up the authority account made by `init_position`, causing a limited DoS. They cannot steal funds or even turn on swapping. However, note that it can also create a position which can then be activated by one of the `dex_admin.auths` authorities.
- **`dex_admin.authority`:** This can only replace itself using `update_authority` and it can change the keys in the admin list using `update_auths`. Its actual power matches whatever the keys in the admin list can do (since it can add itself), except it also has the power to kick keys out of that list.
- **`dex_admin.auths` List:** This is a list of up to 10 keys. Every key on this list can run all gated admin commands described in the next section. Right now, if any single key on this list is compromised,

it can lead to a total loss of funds. The team stated that they ideally plan to split this list into smaller, more specific roles based on the worst-case effect of a key compromise.

The main improvement that is possible is a better differentiation of the risk of the admin instructions controlled by the pubkeys in `dex_admin.auths`. We propose one in the next section.

Admin Instructions Risk Classification

After review, we would recommend the following split of instructions into high-risk and lower-risk instructions which can be executed by the pubkeys in `dex_admin.auths`. High-risk instructions should be handled by a well-protected authority, ideally a multisig. If for whatever reason one of the high-risk instructions has to be executed often or even automatically, great care should be taken to limit the effects of a compromise by other means.

The Fluid Lend team also mentioned that they may want to gate the changing of parameters behind a separate contract that enforces certain rules and possibly timelocks for parameter changes.

Tier 1: High Risk (Can cause loss of funds or needs strict protection)

- `update_center_price_address`: Sets the external price feed (oracle), along with how quickly the price is shifted. If this were compromised, the attacker could simply change the center price, which is obviously critical.
- `update_range_percents`: This shifts the liquidity concentration around the center price, which allows an attacker to warp swap pricing.
- `update_center_price_limits`: Forces prices to cap out if they move outside a specific window, warping how trades are priced.
- `turn_on_smart_debt`: Directly lets the attacker withdraw funds, causing an obvious loss of money.
- `turn_on_smart_col`: Turns on smart collateral trading for a new pool. This doesn't directly steal money, but this is a major system change that should still be put into Tier 1.
- `Unpausing`: While unpausing the protocol (or any of the other more fine-grained pausing knobs) doesn't directly steal money, a paused contract may mean an exploit is actively happening. Restarting the system should be restricted to a highly secure key.
- `update_threshold_percent`: While there's no direct loss-of-funds if compromised, this is part of the pricing logic and should likely still be highly secure.
- `init_position`: This creates an isolated position for a protocol; it still needs to be activated (e.g. for borrowing) to be able to do much. However, note that this can be done using `update_user_borrow_config` and the related instructions, which is why we recommend gating this instruction within Tier 1.

Tier 2: Lower Risk

- `update_fee_and_revenue_cut`: Can be used to make users pay higher fees or stop trades, but an attacker cannot directly steal funds with this instruction.
- Pausing (these are separate instructions for the DEX, users, and swaps and arbitrage): This stops the protocol from working, but the command needs to be easy to use so the team can freeze the system quickly during an exploit.
- `update_utilization_limit`, `update_user_supply_config`, `update_user_withdrawal_limit`: these are simply parameters for lending from the liquidity layer and for the protocol usage, they do not appear critical.
- `update_user_borrow_config`: This sets a protocol's borrow config, much like the parameter updates from the last bullet point. However, note that it should be considered whether it's worthwhile to gate `max_debt_ceiling` separately.
- `init_dex`: This cannot do much until the pool is actually filled with money.
- `update_max_supply_shares` and `update_max_borrow_shares`: Both of these are simply caps on the share count, at worst they can cause a DoS if set too low.

Jupiter keeps a public list of its multisigs and authorities, which also apply to the other Jupiter Lend systems, at https://jup.ag/lend/transparency#multisigs

Protecting Against Authority Compromises

Neodyme also discussed Jupiter Lend's general OpSec stance with several members of their team as it pertains to the DEX contract's authorities.

Of primary importance in protecting against authority compromises is private key hygiene. We recommended an entirely separate device that is used only for signing, with an additional hardware wallet layer to make key extraction harder. We also recommended having sufficiently large subsets of the multisig that use different vendors and frameworks for protecting their keys in order to eliminate single points of failure.

Jupiter stated that they currently have a multi-vendor setup, with most signatories using hardware wallets.

Of course, transactions that are signed must also be verified by each multisig participant. This step is often skipped in multisigs due to capacity constraints and the effort involved, but it's essential. Many past high-profile authority compromises could have been avoided by proper checking of the transactions that were signed.

Jupiter stated that they have a separate internal UI for Squads for non-technical people, as well as custom code for displaying and verifying transaction data, which is not shared between signers. They also stated that for upgrade transactions, the code will be verified both internally and by external auditors.

Monitoring is essential as well, and Jupiter stated that they have it set up for oracle liveness and discrepancies from trusted sources, for whale activity and for multisig actions that are taken. Individual transaction monitoring is not currently set up, but they stated that they are exploring this.

Neodyme also had a long and fruitful discussion with members of the Jupiter team about choice and security of their multisig framework, trusting RPC, key backups, succession planning, break-glass plans and security of communication channels, among other things. The details are omitted in this report, but the auditors would like to emphasize that in almost all domains, the Jupiter team was knowledgeable and aware of best practices.

A | About Neodyme

Security is difficult.

To understand and break complex things, you need a certain type of people. People who thrive in complexity, who love to play around with code, and who don't stop exploring until they fully understand every aspect of it. That's us.

Our team never outsources audits. Having found over 80 High or Critical bugs in Solana's core code itself, we believe that Neodyme hosts the most qualified auditors for Solana programs. We've also found and disclosed critical vulnerabilities in many of Solana's top projects and have responsibly disclosed issues that could have resulted in the theft of over $10B in TVL on the Solana blockchain.

All of our team members have a background in competitive hacking. During such hacking competitions, called CTFs, we competed and collaborated while finding vulnerabilities, breaking encryption, reverse engineering complicated algorithms, and much more. Through the years, many of our team members have won national and international hacking competitions, and keep ranking highly among some of the hardest CTF events worldwide. In 2020, some of our members started experimenting with validators and became active members in the early Solana community. With the prospect of an interesting technical challenge and bug bounties, they quickly encouraged others from our CTF team to look for security issues in Solana. The result was so successful that after reporting several bugs, in 2021, the Solana Foundation contracted us for source code auditing. As a result, Neodyme was born.