Ω

Team Omega

https://teamomega.eth.limo

Almagest Labs DLF Audit

Final Report

September 22, 2025

Summary

Almagest Labs/Inverter has asked Team Omega to audit a set of contracts defining the Digital Liquidity Fund (DLF) token and order manager smart contracts.
The current document is a preliminary audit report. The Inverter team will read this and fix any issues they choose to fix, and Omega will subsequently review the fixes and write a final report.

Team Omega
Team Omega (https://teamomega.eth.limo/) specializes in Smart Contract security audits on the Ethereum ecosystem.

Almagest/Inverter
Inverter (https://www.inverter.network/) is the pioneering Web3 protocol for token economies enabling conditional token issuance, dynamic utility management and token distribution.

Scope of the Audit
The scope is as detailed in the following document:
https://docs.google.com/document/d/1jRuMVPnfRJ9KL9cqsidOvC47TyH47v9W34UYrrKnXe4/edit?tab=t.0

Team Omega audited 2 of the Solidity files in the following repository:

```
https://github.com/InverterNetwork/iTRY-monorepo
```

Specifically, the following files:

- `packages/foundry/contracts/dlf/OrderManager.sol`
- `packages/foundry/contracts/dlf/DLF.sol`

We audited commit `509494f2794b58594830558d0c385c3b99c72dc2`

Methodology

The audit report has been compiled on the basis of the findings from different auditors (Jelle and Ben). The auditors work independently. Each of the auditors has several years of experience in developing smart contracts and web3 applications.

Liability

The audit is on a best-effort basis. In particular, the audit does not represent any guarantee that the software is free of bugs or other issues, nor is it an endorsement by Team Omega of any of the functionality implemented by the software.

Summary of findings

We found **no high or medium severity issues** - these are issues that can lead to a loss of funds, and are essential to fix, or issues we believe you should definitely address. **1** issue was classified as "low", and **5** issues were classified as "info" - we believe the code would improve if these issues were addressed as well.

Severity	**Number of issues**	**Number of resolved issues**
High	0	0
Medium	0	0
Low	1	0
Info	5	2

Resolution

After we delivered a preliminary report, the developers addressed the issues mentioned in the report in the following commit:

```
39542bcdc0780ed20612ecfdca022510cb32f896
```

We have checked the changes and marked the resolution in the issues below.

Findings

General

G1. The signer of the Order has no guarantee that their Order will be executed as intended [info] [not resolved]

The function to create mint and redeem orders implement only one side of the trade on-chain - the buyer/redeemer pays. By design, the subsequent payment happens as a result of an off-chain process. This means that the caller has no guarantee that the trade will be executed on the conditions that they signed for.
Recommendation: Remove the "promise" from the `Order` struct - remove the `dlf_amount` from the `buyOrder` and the `collateral_amount` from the sell order.
Resolution: The issue was not resolved.
Team Response: The separation of payment collection from order fulfillment is an intentional design required by our 24-hour RWA settlement process, which involves manual operations including currency conversion, fund purchases, and order netting. The economic relationships are governed by our Terms & Conditions, validated with launch partners. We are actively working on broker integration to enable binding real-time quotes in future iterations.

G2. Test coverage is incomplete [info] [not resolved]

Test coverage of the contracts is very summary - issue D1 should be caught by a test.
Recommendation: Write more tests, ideally should reach 100% coverage.
Severity: Info
Resolution: The issue was not resolved.

G3. Improper roles management [info] [not resolved]

The contracts are using the roles based access control logic to manage contract functionalities. Yet instead of defining proper role managers, the contracts leave all role management to the default admin role, and add helper functions which allow other roles to also grant and revoke certain roles. While this can work, this is not the proper pattern for using the `AccessControl`, and besides unnecessary contract bloating and gas inefficiency, can lead to unexpected results. For example, in the DLF contract, it is possible for the admin to set an address as both blacklisted and whitelisted at the same time by calling `grantRole` directly. Additionally, functions like `addMinter` and `removeMinter` are completely unnecessary, as the admin could just use `grantRole` and `revokeRole` instead of these.
Recommendation: Remove unnecessary helper functions for granting and revoking roles, and instead set up proper role management in the initializer/ constructor. Also reconsider the use of `AccessControl` roles for managing blacklist and whitelist, as this is not its intended use case, and a simpler solution could be easier and more gas efficient for it.
Severity: Info
Resolution: The issue was not resolved.
Team Response: We have decided to maintain the current implementation, which is functioning correctly with comprehensive test coverage. The helper functions provide important functionality including event emissions and business logic validation beyond simple role management. For this INFO-level finding, the risk of introducing bugs through refactoring outweighs the optimization benefits for us.

DLF.sol

D1. Pre transfer checks are written inefficiently [info] [partially resolved]

The `_beforeTokenTransfer` has checks which are meant to validate and limit token transfers, however, these are written inefficiently for multiple reasons:

- Most of the code between the `FULLY_ENABLED` and `WHITELIST_ENABLED` cases is duplicated, and could be moved to a common call instead of copy pasted.
- Identical storage calls for hasRole are potentially executed multiple times, while it would be more efficient to save the result in memory for hasRole calls whose value may be needed multiple times.
- The most common transfer case - a normal transfer between users, is checked last. This makes normal transfers much more expensive than they should be, as they now must check all other conditions before.

Recommendation: Rewrite the pre-transfer conditions more efficiently.
Severity: Info

Resolution: The issue was partially resolved. The efficiency of the code was improved, but it still has certain inefficiencies. First, the `_isPrivilegedOperation` call could be moved above the blacklist checks instead of being both inside and after them, as the function can stop and return in both cases if action is privileged. Second, there is no need to save the whitelist statuses in memory variables, as they are only used once.

OrderManager.sol

O1. createMintOrderWithApproval and createRedeemOrderWithApproval can be griefed [low] [not resolved]

There is a (theoretical) griefing attack in which the functions `createMintOrderWithApproval` and `createRedeemOrderWithApproval` are frontrun by an attacker who can make these fail consistently. The attack works as follows:

1. minter send tx to mempool to call `createMintOrderWithApproval` and passes `permitParams`
2. attacker intercepts `permitParams` in mempool and executes the permit order on the `collateral_asset`. Which succeeds, the permit is given, and the nonce is used
3. the transaction from the minter now reverts, and the collateral is not transferred.
4. minter cannot execute the same transaction, because the `permitParams` now have a used nonce. To use `createMintOrderWithApproval`, the user must sign a new permit transaction.

Recommendation: The attack is very theoretical, and may not warrant a code change - but one possibility is to skip the call to `executePermit` if enough tokens are approved to execute the transaction. Another way is add code to the backend that will re-execute the order by calling `createMintOrder` (which would work because the permit is already executed)
Severity: Low
Resolution: The issue was not resolved.
Team Response: We acknowledge this theoretical griefing attack vector. The backend does validate if a user has already approved sufficient tokens, in which case it will call the createMintOrder and createRedeemOrder functions directly. Within the current scope, the backend will be setting the transaction to FAILED, given we designed it in a way where it never fails, and the backend doesn't need to track reverted transactions to re-execute orders. We will however consider your suggestion for future iterations.

O2. fx_rate is missing in Event Signature [info] [resolved]

The `BuyOrderCreated` and `SellOrderCreated` events include all order data except for the `fx_rate`.
Recommendation: Add an `fx_rate` field to these events

Severity: Info
Resolution: The issue was resolved as recommended.

O3. Code duplication in createMintOrderWithApproval and createRedeemOrderWithApproval [info] [resolved]

The functions `createMintOrderWithApproval` and `createRedeemOrderWithApproval` implement almost exactly the functionality of their correlated functions without the approval signature. It would be better to deduplicate that code to minimize the chance of errors.
Recommendation: Remove duplicated code.
Severity: Info
Resolution: The issue was resolved as recommended.