SHERLOCK

Security Review For Inverse Finance

Public Audit Contest Prepared For: **Inverse Finance**
Lead Security Expert: **000000**
Date Audited: **November 10 - November 14, 2025**

Introduction

Inverse Finance is the DAO that develops and manages the FiRM fixed-rate lending protocol, and DOLA, its debt-backed, decentralized stablecoin. This contest audits the Junior Tranche insurance mechanism where depositors earn yield while serving as a buffer against bad debt, with particular focus on ERC-4626 accounting under slashing events, withdrawal escrow mechanics, and automated debt repayment logic.

Scope

Repository: sherlock-scoping/InverseFinance__JuniorDola

Audited Commit: 3e5a39251fe92abaa306657c62f9b45ce6c1b234

Final Commit: e85ac217d9019ccb3573401b56b6aa44c6fad719

Files:

- src/FiRMSlashingModule.sol
- src/jDola.sol
- src/LinearInterpolationDelayModel.sol
- src/WithdrawalEscrow.sol

Final Commit Hash

e85ac217d9019ccb3573401b56b6aa44c6fad719

Findings

Each issue has an assigned severity:

- High issues are directly exploitable security vulnerabilities that need to be fixed.
- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.

Issues Found

High	Medium
0	2

Issues Not Fixed and Not Acknowledged

High	Medium
0	0

Security experts who found valid issues

000000
0x23r0
0xAlipede
0xAlix2
0xBoraichoT
0xc0ffEE
0xeix
0xheartcode
0xpetern
0xsh
0xv1bh4
7
A_Failures_True_Power
Artur
Bobai23
CasinoCompiler
Colin
DevBear0411
EV_om
JeRRy0422
JohnTPark24
KaplanLabs
MaratCerby
Olugbenga-ayo
OxSath404
Pataroff
VCeb
al0x23
algiz
bbl4de
dandan
deucefury
future
greekfreakxyz
jayjoshix
klaus
maigadoh
mladenov
newspacexyz
nodesemesta
shaflow01
silver_eth
tobi0x18
touristS
typicalHuman
v_2110
vangrim
vinica_boy
y4y
yovchev_yoan

Issue M-1: ERC4626 maxDeposit() Violates Standard by Not Enforcing Actual Deposit Limits

Source: https://github.com/sherlock-audit/2025-11-inverse-finance-junior-tranche-judging/issues/20

Found by

000000, 0x23r0, 0xAlipede, 0xBoraichoT, 0xc0ffEE, 0xeix, 0xheartcode, 0xpetern, 0xsh, A_Failures_True_Power, Artur, Bobai23, CasinoCompiler, Colin, DevBear0411, EV_om, JeRRy0422, JohnTPark24, KaplanLabs, Olugbenga-ayo, Pataroff, VCeb, al0x23, algiz, bbl4de, dandan, future, greekfreakxyz, jayjoshix, maigadoh, mladenov, newspacexyz, nodesemesta, shaflow01, silver_eth, tobi0x18, touristS, typicalHuman, v_2110, vangrim, vinica_boy, y4y, yovchev_yoan

Summary

The jDola contract violates ERC-4626's "MUST" requirement for `maxDeposit()` by returning `type(uint256).max` instead of the actual maximum that can be deposited without revert. The contract has multiple deposit limits (MIN_SHARES, MAX_ASSETS, MAX_SHARES) that are not reflected in `maxDeposit()`, causing potential integration failures.

https://eips.ethereum.org/EIPS/eip-4626

Vulnerability Detail

According to ERC-4626, `maxDeposit()` has specific requirements:

> **MUST return the maximum amount of assets deposit would allow to be deposited for receiver and not cause a revert, which MUST NOT be higher than the actual maximum that would be accepted**

However, jDola inherits Solmate's default implementation: https://github.com/transmissions11/solmate/blob/main/src/tokens/ERC4626.sol#L160-L162

```
function maxDeposit(address) public view virtual returns (uint256) {
    return type(uint256).max;  // Always returns unlimited
}
```

But jDola enforces multiple limits in `afterDeposit()`:

https://github.com/sherlock-audit/2025-11-inverse-finance-junior-tranche/blob/main/InverseFinance__JuniorDola/src/jDola.sol#L96-L100

```
function afterDeposit(uint256, uint256) internal override {
    require(totalSupply >= MIN_SHARES, "Shares below MIN_SHARES");        // 1e18
    ↪  minimum
    require(totalSupply <= MAX_SHARES, "Shares above MAX_SHARES");        //
    ↪  2^128-1 maximum
    require(totalAssets() >= MIN_ASSETS, "Assets below MIN_ASSETS");      // 1e18
    ↪  minimum
}
```

ERC-4626 Violation Examples:

1. **MAX_SHARES Limit**: If `totalSupply = 2128 - 1000`, `maxDeposit()` returns unlimited but any deposit would revert
2. **MAX_ASSETS Limit**: `totalAssets()` calculation caps at `MAX_ASSETS = 1e32`, but this isn't reflected

Impact

Breaks core EIP-4626 functionality requirements and explicit violation of ”MUST” requirement

Tool used

Manual Review

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/InverseFinance/JuniorDola/pull/11

Issue M-2: Off-by-one error in exit window check allows users to avoid the withdrawal fee

Source: https://github.com/sherlock-audit/2025-11-inverse-finance-junior-tranche-judging/issues/318

Found by

000000, 0xAlipede, 0xAlix2, 0xv1bh4, 7, Colin, MaratCerby, OxSath404, deucefury, klaus, maigadoh, silver_eth

Summary

A boundary check in `WithdrawalEscrow.queueWithdrawal` allows a user to avoid the intended full withdrawal fee by renewing exactly at the start of their exit window. Because the code checks `block.timestamp > exitWindowStart` (strict) instead of `block.timestamp >= exitWindowStart`, a renewal at the exact start timestamp charges the fee only on the newly queued amount, instead of the whole withdrawal amount.

Root Cause

In `queueWithdrawal` https://github.com/sherlock-audit/2025-11-inverse-finance-junior-tranche/blob/main/InverseFinance__JuniorDola/src/WithdrawalEscrow.sol#L91-L97:

```
if(withdrawFeeBps > 0){
    //If user has had a chance to withdraw, we apply full fee, otherwise only apply
    ↪  fee on new amount
    fee = totalWithdrawAmount > amount && block.timestamp > exitWindowStart ?
        totalWithdrawAmount * withdrawFeeBps / 10000 :
        amount * withdrawFeeBps / 10000;
    totalWithdrawAmount -= fee;
}
```

At equality (`block.timestamp == exitWindowStart`), the “full-fee” branch is not taken.

Internal Pre-conditions

1. `withdrawFeeBps > 0`.
2. Caller already has a queued withdrawal (`withdrawAmounts[msg.sender] > 0` and a nonzero `exitWindows[msg.sender].start`).

External Pre-conditions

None.

Attack Path

1. User queues an initial withdrawal `A` and receives a window `[start = S, end]`.
2. At `t = S`, call `queueWithdrawal(0, maxWithdrawDelay)` to renew.
3. Because `block.timestamp > S` is false at `t = S`, the fee is applied to amount (0) instead of `totalWithdrawAmount`, resulting in zero fee.
4. A new later window is set while the outstanding amount remains intact and uncharged. Repeat as needed.

Note that the `amount` in the above attack path example does not need to be 0; it can also be non-zero. Following the above attack path, fees are only charged on the new amount.

Though "likelihood is not considered when identifying the severity and the validity of the report" as per Sherlock rules, it's still worth mentioning that likelihood can be increased by some methods:

- The renew call can be reliably targeted with a guard (e.g., a contract requiring `block.timestamp == S` and reverting otherwise), so an inclusion with the wrong timestamp reverts and does not accidentally trigger the full-fee branch.
- The start timestamp `S` is also predictable and targets the timestamp boundary on PoS Ethereum, which uses fixed 12□second slots.

Impact

- The intended withdrawal fee on the full outstanding amount is not collected, reducing the donation to the vault and harming remaining shareholders' yield.
- The attack can be repeated for each new withdrawal window to continuously avoid fees.

A concrete example:

- `withdrawFeeBps = 100` (1%), vault TVL □ $10,000,000.
- User already has withdrawAmounts = 5,000,000 shares at $1/share (□ $5,000,000 assets), which was initialized previously.
- User renews withdrawal with an additional 0 amount at the start timestamp of the withdrawal window.
- Intended fee in the renewal: 5,000,000 × 1% = 50,000 shares □ $50,000 donated to the vault.
- Actual fee charged is $0, so the remaining stakers lose □ $50,000 in expected fee□derived yield (which is a 100% loss of the intended fee on this renewal).

PoC

No response

Mitigation

Replace `block.timestamp > exitWindowStart` with `block.timestamp >= exitWindowStart`.

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/InverseFinance/JuniorDola/pull/11

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.