SHERLOCK

Security Review For Inverse Finance

Collaborative Audit Prepared For:	Inverse Finance
Lead Security Expert(s):	EgisSecurity
	juan
	Spearmint
Date Audited:	October 13 - October 16, 2025

Introduction

Inverse Finance is the decentralized autonomous organization that develops and manages the FiRM fixed rate lending protocol, DOLA, its debt-backed, decentralized stablecoin, and sDOLA, the yield-bearing version of DOLA.

Scope

Repository: sherlock-scoping/InverseFinance__JuniorDola

Audited Commit: 2bae3fa7ae6d88e808ef73baf73d305e0b67dd20

Final Commit: 41af61ea57de928cb3a706379282c3850a9c7136

Files:

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

Final Commit Hash

41af61ea57de928cb3a706379282c3850a9c7136

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.
- Low/Info issues are non-exploitable, informational findings that do not pose a security risk or impact the system's integrity. These issues are typically cosmetic or related to compliance requirements, and are not considered a priority for remediation.

Issues Found

High	Medium	Low/Info
1	1	3

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue H-1: Anyone can steal all jDola from `WithdrawalEscrow` [RESOLVED]

Source: https://github.com/sherlock-audit/2025-10-inverse-finance-oct-13th/issues/5

Summary

The withdraw escrow trusts any vault address passed to `queueWithdrawal` and treats it as an ERC4626 without validating it is a known/benign implementation.

Vulnerability Detail

When `withdrawFeeBps > 0`, the escrow executes a fee flow that:

1. calls `_vault.redeem(fee, address(this), address(this))` on the untrusted vault and uses the returned value as `amount` to a `ERC20.approve` call
2. reads `_vault.asset()` (also controlled by the untrusted vault),
3. approves the amount from 1. returned token to the untrusted vault for `dolaRedeemed`

Because the escrow:

- does not whitelist vaults,
- does not bind a vault to a pre-verified asset,
- uses an approve-then-pull pattern to an untrusted contract right after external calls, the attacker can drain escrow-held tokens (e.g., user shares waiting to withdraw) in a single call when fees are enabled.

PoC https://gist.github.com/NicolaMirchev/635376aafae7e1205d4f1b6ba542d139

Impact

Theft of all `jDola` tokens in `withdrawEscrow`

Code Snippet

https://github.com/sherlock-audit/2025-10-inverse-finance-oct-13th/blob/dd8b79451181409793a3f85da0a75d37dff7598d/InverseFinance__JuniorDola/src/WithdrawalEscrow.sol#L101-L103

Tool Used

Manual Review

Recommendation

Implement a vault whitelisting

Issue M-1: `queueWithdrawal` redeem won't work with `amount = 0` and `block.timestamp <= exitWindowStart` [RESOLVED]

Source: https://github.com/sherlock-audit/2025-10-inverse-finance-oct-13th/issues/6

Summary

`queueWithdrawal` redeem won't work with `amount = 0` and `block.timestamp <= exitWindowStart`

Vulnerability Detail

Users can renew their withdrawals by calling `queueWithdrawal` with `amount = 0`, this coment states this

```
 //To renew a withdrawal, queue a 0 amount withdrawal
    function queueWithdrawal(address vault, uint amount) external nonReentrant {
...
```

The issue is that if `withdrawFeeBps > 0` then a fee will be applied.

```
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;
       }
```

If a user is trying to renew his withdraw, then `totalWithdrawAmount > amount` will always be true, since he already has a queued withdraw and `block.timestamp > exitWindowStart` in this case will be false, he is trying to renew his window prior to his window's start.

In this the fee is applied to `amount`, since `amount = 0` no fee is applied.

The issue is when `fee` is attempted to be redeemed.

```
if(withdrawFeeBps > 0){
           //@lead can potentially `reedem` 0 here, which will fail
           uint dolaRedeemed = _vault.redeem(fee, address(this), address(this));
           _vault.asset().approve(vault, dolaRedeemed);
           _vault.donate(dolaRedeemed);
       }
```

Redeeming 0 is impossible, because of how `redeem` works.

```
function redeem(
        uint256 shares,
        address receiver,
        address owner
    ) public virtual returns (uint256 assets) {
        if (msg.sender != owner) {
            uint256 allowed = allowance[owner][msg.sender]; // Saves gas for
            ↪  limited approvals.

            if (allowed != type(uint256).max) allowance[owner][msg.sender] =
            ↪  allowed - shares;
        }

        // Check for rounding error since we round down in previewRedeem.
        require((assets = previewRedeem(shares)) != 0, "ZERO_ASSETS");
```

`previewRedeem` does the following

```
function convertToAssets(uint256 shares) public view virtual returns (uint256) {
        uint256 supply = totalSupply; // Saves an extra SLOAD if totalSupply is
        ↪  non-zero.

        return supply == 0 ? shares : shares.mulDivDown(totalAssets(), supply);
    }
```

`0` multiplied by something then divided by something is always 0, so `previewRedeem` returns 0 and the tx reverts. This punishes users, as they can't renew their window prior to their current window's start, thus they are always forced to pay a fee for the second time, it also breaks the invariant of letting users renew their window whenever possible.

Impact

Renewing queue withdrawals doesn't work as intented

Code Snippet

https://github.com/sherlock-audit/2025-10-inverse-finance-oct-13th/blob/dd8b79451181409793a3f85da0a75d37dff7598d/InverseFinance__JuniorDola/src/WithdrawalEscrow.sol#L99-L103

Tool Used

Manual Review

Recommendation

Change the fee redemption to the following

```
if(withdrawFeeBps > 0 && fee > 0){
            uint dolaRedeemed = _vault.redeem(fee, address(this), address(this));
            _vault.asset().approve(vault, dolaRedeemed);
            _vault.donate(dolaRedeemed);
        }
```

Extra safe would be like so.

```
uint preview = _vault.previewRedeem(fee);
if (withdrawFeeBps > 0 && fee > 0 && preview > 0) { ... }
```

This way any possible to 0 rounding will also be handled and will allow users to queue.

Issue L-1: Broken invariant `totalAssets() < MIN_ASSETS` in `jDola::slash` [RESOLVED]

Source: https://github.com/sherlock-audit/2025-10-inverse-finance-oct-13th/issues/7

Summary

Basically the invariant will not hold if everything is slashed (first `if` branch):

```
function slash(uint amount) external onlySlashingModule() returns(uint) {
    //Make sure slashed amount doesn't exceed total supply
    //TODO: Add logic to handle still accruing revenue
    if(totalAssets() < amount){
        amount = totalAssets(); // @sus this may result in breaking the invariant
         ↪  `totalAssets() < MIN_ASSETS`
    //Make sure slashed amount doesn't leave junior tranche with less assets than
    ↪  MIN_ASSETS
    //TODO: Consider allowing 0 assets
    }
```

And then prev week revenue is accumulated such that the amount is < `MIN_ASSETS`:

```
function totalAssets() public view override returns (uint) { // @ok
    uint week = block.timestamp / 7 days;
    uint timeElapsed = block.timestamp % 7 days;
    uint remainingLastRevenue = weeklyRevenue[week - 1] * (7 days - timeElapsed) /
    ↪  7 days;
    uint actualAssets = asset.balanceOf(address(this)) - remainingLastRevenue -
    ↪  weeklyRevenue[week];
    return actualAssets < MAX_ASSETS ? actualAssets : MAX_ASSETS;
}
```

It is also reachable if we shash eveything (we don't enter `if`, `if else`) and then `totalAssets` increases just below the `MIN_ASSETS` because of the prev weekly distribution

Impact

Having `totalAssets() > 0 && totalAssets() < MIN_ASSETS`

Code Snippet

https://github.com/sherlock-audit/2025-10-inverse-finance-oct-13th/blob/dd8b79451181409793a3f85da0a75d37dff7598d/InverseFinance__JuniorDola/src/jDola.sol#L176-L186

Tool Used

Manual Review

Recommendation

Ensure that the invariant holds. If `weeklyRevenue[last week] >= 1d18`, distribute the amount of 1e18 instantly and withdraw it from `weeklyRevenue[last week]`, if we are slashing all the assets

Discussion

08xmt

Decided it's safer to make MIN_ASSETS + remaining revenue unslashable. In any given week this is unlikely to be an impactful amount.

Issue L-2: Lack of setOperator function in the jDola contract [RESOLVED]

Source: https://github.com/sherlock-audit/2025-10-inverse-finance-oct-13th/issues/8

Summary

The `jDola` contract does not have a function to change the operator address after deployment.

As a result the operator's address can never be updated post deployment, even in emergency scenarios (for example if the operator turns malicious)

Recommendation

Consider adding a `setOperator(address _operator)` function with the `onlyGov` modifier to allow governance to update the operator role.

Issue L-3: Users cannot specify a maximum withdraw delay when withdrawing [RESOLVED]

Source: https://github.com/sherlock-audit/2025-10-inverse-finance-oct-13th/issues/9

Summary

The `WithdrawalEscrow` contract determines each user's withdrawal delay dynamically through the `withdrawDelayModel`, without allowing users to specify their own maximum acceptable delay.

Vulnerability Detail

When multiple users queue withdrawals at the same time, those whose transactions are processed later may receive unexpectedly significantly longer withdrawal delays than the earlier users. Since the contract does not allow users to set a maximum acceptable delay, their transactions will still execute even if the resulting lockup period becomes unexpectedly long.

Impact

Under specific conditions (high withdrawal activity in a short period of time), some users may face unexpectedly long lockups

Code Snippet

https://github.com/sherlock-audit/2025-10-inverse-finance-oct-13th/blob/dd8b79451181409793a3f85da0a75d37dff7598d/InverseFinance__JuniorDola/src/WithdrawalEscrow.sol#L65

Tool Used

Manual Review

Recommendation

Consider adding a parameter that allows users to specify a maximum acceptable withdrawal delay, and revert the transaction if the calculated delay exceeds this limit.

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.