yAudit Inverse Finance Dola savings Review

Review Resources:

Code repositories and documentation were used during this audit.

Auditors:

- adriro
- pandadefi

Table of Contents

Review Summary

Dola Savings

DolaSavings is a staking platform allowing users to earn rewards by depositing DOLA tokens. It aims to promote long-term holding by distributing DBR tokens based on the duration and amount of DOLA staked.

The contracts of the Dola savings Repo were reviewed over 3 days. The code review was performed by 2 auditors between January 4th and January 7th, 2024. The repository was under active development during the review, but the review was limited to the latest commit at the start of the review. This was commit 5c38feed71ef71425ecd6b121574220e94ab8f8d for the Dola savings repo.

Scope

The scope of the review consisted of the following contracts at the specific commit:

- src/DolaSavings.sol
- src/sDola.sol
- src/sDolaHelper.sol

After the findings were presented to the Dola savings team, fixes were made and included in several PRs.

This review is a code review to identify potential vulnerabilities in the code. The reviewers did not investigate security practices or operational security and assumed that privileged accounts could be trusted. The reviewers did not evaluate the security of the code relative to a standard or specification. The review may not have identified all potential attack vectors or areas of vulnerability.

yAudit and the auditors make no warranties regarding the security of the code and do not warrant that the code is free from defects. yAudit and the auditors do not represent nor imply to third parties that the code has been audited nor that the code is free from defects. By deploying or using the code, Inverse Finance and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	Follows standard practices.
Mathematics	Good	Calculations are accurate with proper overflow checks.
Complexity	Good	Code is well-organized and modular.
Libraries	Good	Uses well-tested libraries without modifications.
Decentralization	Good	User funds are safe from governance actions.
Code stability	Good	Stable with no known issues in the current environment.
Documentation	Low	Functions are lacking NatSpec comments.
Monitoring	Low	Missing events on state variable changes.
Testing and verification	Average	Adequate tests cover major functionalities.

Findings Explanation

Findings are broken down into sections by their respective impact:

- Critical, High, Medium, Low impact
  - These are findings that range from attacks that may cause loss of funds, impact control/ownership of the contracts, or cause any unintended consequences/actions that are outside the scope of the requirements.
- Gas savings
  - Findings that can improve the gas efficiency of the contracts.
- Informational
  - Findings including recommendations and best practices.

High Findings

1. High - sDola vault is susceptible to the inflation attack

The first depositor in the sDola.sol contract can inflate the value of a share to cause rounding issues in subsequent deposits.

Technical Details

The sDola ERC4626 vault is susceptible to a vulnerability known as the *Inflation Attack*, in which the first depositor can be front-run by an attacker to steal their deposit.

Let’s imagine a user wants to deposit X amount of DOLA in sDola.

1. The attacker deposits 1 wei of DOLA in sDola, they own 1 share of sDOLA.
2. The attacker stakes `X / 2` DOLA in DolaSaving on behalf of the sDola vault, now total assets in sDola are `X / 2 + 1`.
3. The user deposit transaction goes through, they are minted `roundDown( X * 1 / (X / 2 + 1) ) = 1` share.
4. The attacker redeems their share of sDOLA and receives `(X + X / 2) / 2 = 3/4 * X`. Their profit is `3/4 * X - X / 2 - 1 = X / 4 - 1`.

Impact

High. An attacker can steal part of the initial deposit in the vault.

Recommendation

There are different ways to mitigate this attack. One of the simplest alternatives is to mint an initial set of dead shares when the vault is deployed so that the attack would become impractical to perform.

Developer Response

Addressed in https://github.com/InverseFinance/dola-savings/pull/9/files.

2. High - `sDola` should not be allowed to be borrowed in a lending borrowing market

The `sDola` price can be manipulated with deposits to `DolaSavings` on the behalf of `sDola` contract.

Technical Details

When an asset whose price can be manipulated atomically is used as collateral and borrowed, the lending market is at risk. If a large deposit is made to `DolaSavings` in the name of the `sDola` contract, it artificially inflates the value of sDola. This can lead to a scenario where the borrower can borrow more than the actual collateral value. See: cream finance hack

Impact

High. sDola can't be borrowed.

Recommendation

Document the issue, and make sure protocol integrators are aware of the pitfalls of using `sDola`.

Developer Response

Addressed in https://github.com/InverseFinance/dola-savings/pull/8/commits/0c8f83a4afa5cb25513ed74060cf369ddd55d982.

Low Findings

1. Low - Consider implementing two-step procedure for updating protocol addresses

A copy-paste error or a typo may end up bricking protocol operability.

Technical Details

The `gov` state variable is key to the protocol governance.

```
71 | function setGov(address _gov) public onlyGov { gov = _gov; }
```

DolaSavings.sol#L71

```
100 | function setGov(address _gov) external onlyGov {
101 |         gov = _gov;
102 |     }
```

sDola.sol#L100

Impact

Low. Uploading protocol governance needs to be done with extra care.

Recommendation

Add a two-step governance address update.

Developer Response

Addressed in https://github.com/InverseFinance/dola-savings/pull/4.

2. Low - Missing checks for `address(0)` on `stake()` `recipient`

Funds can be staked by mistake to the `address(0)`.

Technical Details

```
90|    function stake(uint amount, address recipient) public updateIndex(recipient) {
91|        balanceOf[recipient] += amount;
92|        totalSupply += amount;
93|        dola.transferFrom(msg.sender, address(this), amount);
94|    }
```

DolaSavings.sol#L91

Impact

Low. Funds can be lost.

Recommendation

Add a check to make sure the recipient isn't `address(0)`.

Developer Response

Addressed https://github.com/InverseFinance/dola-savings/pull/4.

3. Low - `buyDBR()` call with incorrect `exactDbrOut` might lead to overpaying for `dbr`

With `buyDBR()` taking `exactDolaIn` and `exactDbrOut` as parameters, it's possible that a user misused `exactDbrOut` is not ideal at the moment the transaction is mined. This will have the user paying extra `DolaIn`.

Technical Details

The `exactDbrOut` amount might be different from the ideal amount because of changes on chain or a mistake from the user.

sDola.sol#L88-L98

Impact

Low. Users should use the helper contracts.

Recommendation

Document the existence of the helper contract for users to interact with.

Developer Response

4. Low - Incorrect overflow check in `maxYearlyRewardBudget`

The check in `setMaxYearlyRewardBudget()` is presumably incorrect as the associated comment reads:

> cannot overflow and revert within 10,000 years

Technical Details

Accrued rewards are calculated in `updateIndex` according to the following formula:

```
36:                 uint maxBudget = maxRewardPerDolaMantissa * totalSupply / mantissa;
37:                 uint budget = yearlyRewardBudget > maxBudget ? maxBudget :
yearlyRewardBudget;
38:                 uint rewardsAccrued = deltaT * budget * mantissa / 365 days;
```

Line 38 will overflow if `deltaT * budget * mantissa > 2**256 - 1`, hence we need `budget < 2**256 - 1 / (deltaT * mantissa)`.

If the intention is to support up to 10 years, then the check in `setMaxYearlyRewardBudget()` should be `_max < type(uint).max / (365 days * 10 * mantissa)`.

Impact

Low.

Recommendation

Adjust the overflow check in `setMaxYearlyRewardBudget()`.

Developer Response

Addressed in https://github.com/InverseFinance/dola-savings/pull/5.

5. Low - Missing `sweep()` function as part of `sDola` contract

The `sDola` contract doesn't have a `sweep()` function.

Technical Details

Unlike `DolaSaving`, it's not possible to recover tokens sent by mistake due to the lack of a `sweep()` function. The contract should only have `dbr` tokens; other tokens should be recoverable by the governance multisig account.

Impact

Low. Funds sent by mistake would be lost.

Recommendation

```
+    function sweep(address token, uint amount, address to) public onlyGov {
+        require(address(dbr) != token, "Not authorized");
+        IERC20(token).transfer(to, amount);
+    }
```

Developer Response

Addressed in https://github.com/InverseFinance/dola-savings/pull/6.

Gas Saving Findings

1. Gas - Unnecessary call to `getDbrReserve()` in `buyDBR()`

`getDbrReserve()` calls the saving contract on the `claimable()` function. A claim to the saving contract is done right before the call to `claimable()`, which will then always return zero.

Technical Details

```
89|        savings.claim(address(this));
90|        uint dolaReserve = getDolaReserve() + exactDolaIn;
91|        uint dbrReserve = getDbrReserve() - exactDbrOut;
```

sDola.sol#L89-L91

Impact

Gas savings.

Recommendation

Replace line 91 by

```diff
-         uint dbrReserve = getDbrReserve() - exactDbrOut;
+         uint dbrReserve = dbr.balanceOf(address(this)) - exactDbrOut;
```

Since `getDolaReserve()` is also calling `getDbrReserve()` it is also possible to save even more gas with the following code:

```diff
-        uint dolaReserve = getDolaReserve() + exactDolaIn;
-        uint dbrReserve = getDbrReserve() - exactDbrOut;
+        uint balance = dbr.balanceOf(address(this));
+        uint dolaReserve = getK() / balance + exactDolaIn;
+        uint dbrReserve = dbr.balanceOf(address(this)) - exactDbrOut;
```

To save even more gas, with these changes, you could cache `getK()` instead of calling it twice.

Developer Response

Addressed in https://github.com/InverseFinance/dola-savings/pull/3/commits/7c7683c7bdc6e5b6533b5f002cb853a2fa0d79ba.

2. Gas - Cache storage variables in reward calculation logic

Several storage variables are read multiple times in the implementation of the `updateIndex` modifier and the `claimable()` function.

Technical Details

The following variables are fetched from storage multiple times:

- `yearlyRewardBudget`
- `totalSupply`
- `rewardIndexMantissa`

Impact

Gas savings.

Recommendation

Consider using a local variable as a cache to prevent multiple reads from storage.

Developer Response

Addressed in https://github.com/InverseFinance/dola-savings/pull/3/commits/b77a420808fb92031010fc5122c4b1b63f37b729

3. Gas - In `getDolaReserve()` add an option to pass `getDbrReserve()`

`getDolaReserve()` and `getDbrReserve()` are often called within the same scope, with `getDolaReserve()` making a call to `getDbrReserve()`, it's possible to save gas by passing the `getDbrReserve()` result to `getDolaReserve()`.

Technical Details

```
    function getDbrOut(uint dolaIn) public view returns (uint dbrOut) {
        require(dolaIn > 0, "dolaIn must be positive");
        uint dolaReserve = sDola.getDolaReserve();
        uint dbrReserve = sDola.getDbrReserve();
```

Here we can see `getDolaReserve()` and `getDbrReserve()` are used in the same scope.

```
File: sDola.sol
    function getDolaReserve() public view returns (uint) {
        return getK() / getDbrReserve();
    }

    function getDbrReserve() public view returns (uint) {
        return dbr.balanceOf(address(this)) + savings.claimable(address(this));
    }
```

The `getDbrReserve()` result can be passed to `getDolaReserve()` to prevent additional calls to `balanceOf()` and `claimable()` methods.

sDola.sol#L73-L79

sDolaHelper.sol#L29-L36

Impact

Gas savings.

Recommendation

```
    function getDolaReserve() public view returns (uint) {
        return getK() / getDbrReserve();
    }

+   function getDolaReserve(dbrReserve) public view returns (uint) {
+       return getK() / dbrReserve;
+   }
```

With that added it’s possible to update the helper contract functions getDbrOut() and getDolaIn().

Developer Response

Addressed in https://github.com/InverseFinance/dola-savings/pull/3/commits/1f01bb0cc94e359830b5e44b7c299280ec0d4bf5.

4. Gas - Week elapsed time calculation can be simplified

The elapsed seconds in the current week can be calculated using the modulo operator.

Technical Details

In `totalAssets()`, the `timeElapsed` variable can be simplified as `block.timestamp % 7 days`.

Impact

Gas savings.

Recommendation

```
-   uint timeElapsed = block.timestamp - (week * 7 days);
+   uint timeElapsed = block.timestamp % 7 days;
```

Developer Response

Addressed in https://github.com/InverseFinance/dola-savings/pull/3/commits/45c4d9ff98df5c97952d72af87ab842aaa37c01e.

Informational Findings

1. Informational - Missing limits when setting max amounts

There is one missing limit in `setMaxRewardPerDolaMantissa()`, and this could lead to unexpected scenarios.

Technical Details

```
81 | function setMaxRewardPerDolaMantissa(uint _max) public onlyGov updateIndex(msg.sender) {
82 |         maxRewardPerDolaMantissa = _max;
83 |     }
```

DolaSavings.sol#L81

Impact

Informational.

Recommendation

Consider adding a max limit check.

Developer Response

Addressed in https://github.com/InverseFinance/dola-savings/pull/7/commits/707358b957cf142e896140beb59878f4e999bdc7.

2. Informational - Missing event for a critical parameter change

It is recommended to emit events when updating state variables.

Technical Details

The following functions are missing event emission:

DolaSavings.sol#70 DolaSavings.sol#71 DolaSavings.sol#73 DolaSavings.sol#81 DolaSavings.sol#85 Dola.sol#L81 Dola.sol#L81100

Impact

Informational.

Recommendation

Add events to log the state variable changes.

Developer Response

Partially addressed in https://github.com/InverseFinance/dola-savings/pull/7/commits/46afe0b1350346fc8001bb43da441de1cfb5d70c.

3. Informational - `public` functions not called by the contract should be declared `external` instead

Using external visibility is recommended for clarity.

Technical Details

```
70 | function setOperator(address _operator) public onlyGov { operator = _operator; }

71 | function setGov(address _gov) public onlyGov { gov = _gov; }

73 | function setMaxYearlyRewardBudget(uint _max) public onlyGov updateIndex(msg.sender)
{

81 | function setMaxRewardPerDolaMantissa(uint _max) public onlyGov
updateIndex(msg.sender) {

85 | function setYearlyRewardBudget(uint _yearlyRewardBudget) public onlyOperator
updateIndex(msg.sender) {

90 | function stake(uint amount, address recipient) public updateIndex(recipient) {

96 | function unstake(uint amount) public updateIndex(msg.sender) {

102 | function claimable(address user) public view returns(uint) {

114 | function claim(address to) public updateIndex(msg.sender) {

119 | function sweep(address token, uint amount, address to) public onlyGov {
```

DolaSavings.sol#L70 DolaSavings.sol#L71 DolaSavings.sol#L73 DolaSavings.sol#L81
DolaSavings.sol#L85 DolaSavings.sol#L90 DolaSavings.sol#L96 DolaSavings.sol#L102
DolaSavings.sol#L114 DolaSavings.sol#L119

Impact

Informational.

Recommendation

Change the function visibility.

Developer Response

Addressed in https://github.com/InverseFinance/dola-savings/pull/7/commits/ade8b1d034e9b354dbb647d8d5bc54e1c60728c7.

4. Informational - `else` block unnecessary

By eliminating the `else` block and directly returning the values from the `if`-block, one level of nesting can be removed:

Technical Details

```
64 |        if(timeElapsed > duration) {
65 |             return targetK;
66 |         } else {
67 |             uint targetWeight = timeElapsed;
68 |             uint prevWeight = duration - timeElapsed;
69 |             return (prevK * prevWeight + targetK * targetWeight) / duration;
70 |         }
```

sDola.sol#L64

Impact

Informational.

Recommendation

Code can be replaced by:

```
        if(timeElapsed > duration) {
             return targetK;
         }
        uint targetWeight = timeElapsed;
        uint prevWeight = duration - timeElapsed;
        return (prevK * prevWeight + targetK * targetWeight) / duration;
```

Developer Response

Addressed in https://github.com/InverseFinance/dola-savings/pull/7/commits/85938f54759d950a4f4db045ce3b143ffa971185.

Final remarks

The yAudit of Inverse Finance's Dola Savings platform, conducted by adriro and pandadefi, provided a thorough examination of its smart contracts. The audit, spanning three days, uncovered a range of findings from high to low impact, alongside gas-saving and informational insights. Critical vulnerabilities, such as the susceptibility of the sDola vault to inflation attacks and the potential manipulation of sDola in lending-borrowing markets, were promptly addressed. Lower-impact issues, focusing on aspects like checks and function optimizations, were also noted for improvement. The audit emphasizes the platform's strong foundation in smart contract development and its commitment to security, efficiency, and continuous improvement.