ISPO

Table of contents

Informational	Permit mechanic
	Insufficient zero checks
	Gas optimization: Redundant expressions/variables
	Misuse of input amounts instead of final ones
	Gas optimization: Cache storage variables
	View function not view in the interface
	Redundant variables
	Invalid user.shares calculation
	Unnecessary check
	Incorrect variable name

1. Project Brief

Title	Description
Client	Dega
Project name	lSPO
Timeline	08-01-2024 - 17-01-2024
Initial commit	dd24eb6b922eb055f89614b80bc6cc8e22e708
Final commit	d58a7843afd5e9a378faa9550d55b35e4a56d8

Short Overview

An lSPO is a new way for early adopters to support a project using a blockchain's proof of s
purchase of a token sale like an ICO, participants will delegate tokens (stETH) to an DEGA IS
protected by the LIDO.

Project Scope

The audit covered the following files:

DegalSPO.sol

2. Finding Severity breakdown

All vulnerabilities discovered during the audit are classified based on their potential severity

classification:

Severity	Description
Critical	Bugs leading to assets theft, fund access locking, or any other loss party.
High	Bugs that can trigger a contract failure. Further recovery is possibl the contract state or replacement.
Medium	Bugs that can break the intended contract logic or expose it to DoS loss of funds.
Informational	Bugs that do not have a significant immediate impact and could be

Based on the feedback received from the Customer regarding the list of findings discovered

assigned the following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no lo
Acknowledged	The Customer is aware of the finding. Recommendations for the fir in the future.

3. Summary of findings

Severity	# of Findings
Critical	3 (3 fixed, 0 acknowledged)
High	3 (3 fixed, 0 acknowledged)
Medium	2 (0 fixed, 2 acknowledged)
Informational	15 (13 fixed, 2 acknowledged)
Total	23 (19 fixed, 4 acknowledged)

4. Conclusion

During the audit of the codebase, 19 issues were found in total:

- 3 critical severity issues (3 fixed)
- 3 high severity issues (3 fixed)
- 2 medium severity issues (2 acknowledged)
- 15 informational severity issues (13 fixed, 2 acknowledged)

The final reviewed commit is d58a7843afd5e9a378faa9550d55b35e4a56d841

5. Findings report

CRITICAL-01 | Admin can't withdraw the rewards in full

Description

Line: DegalSPO.sol#L94

Admin passes the stETH amount for withdrawal in the **DegalSPO::adminWithdraw** function decreases **degaTreasuryShares** by the stETH amount.

Impact: Admin can't withdraw all the rewards accrued for the Dega treasury. Part of the rew without the possibility of withdrawal.

Recommendation

We recommend decreasing the **degaTreasuryShares** by the quantity of withdrawn shares

```
degaTreasuryShares -= sharesToWithdraw;
```

CRITICAL-02 | Users can't withdraw stETH in an emergency

Description

Lines: DegalSPO.sol#L236-L239

The vulnerability in the **emergencyWithdraw** function arises from the calculation of **curren** less than **totalStakeTokenDeposited**, the division results in **0** because Solidity does not ha division. This means **currentAmount** will always be **0** in such cases, leading to the require the transaction, thereby preventing any withdrawals.

Impact: This bug renders the emergency withdrawal feature unusable when a user's stake amount, which is a common scenario.

Recommendation

We recommend reordering the operations to perform multiplication before division.

In Code:

CRITICAL-03	Possible underflow

Description

Line: **DegalSPO.sol#L219**

```
totalSharesDeposited -= user.shares;
```

This line can lead to underflow because **user.shares** may be greater than **totalSharesDepo**
Possible scenario:

```
// user1 calls deposit(10): 10 stEth ~ 10 shares
user.amount = 10
user.shares = 10
totalStakeTokenDeposited = 10
totalSharesDeposited = 10
poolETHSize = 10

// +10% rebase: 11 stEth ~ 10 shares
// anyone call assignRewards()
rewardStInt = 11 - 10 = 1
sharesToAssignRewards ~= 0.91
totalSharesDeposited ~= 10 - 0.91 = 9.09
degaTreasuryShares = 0 + 0.91 = 0.91

// admin calls pause() for any reason

// user1 calls emergencyWithdrawal()
pooledEth = lidoContract.getPooledEthByShares(9.09) ~= 10
currentAmount = 10 * 10 / 10 = 10
sharesToWithdraw = lidoContract.getSharesByPooledEth(10) = 9.09

totalStakeTokenDeposited -= 10 = 10 - 10 = 0
totalSharesDeposited -= user.shares = 9.09 - 10 ?? Underflow
```

This happens because **user.shares** doesn't subtract shares, that were transferred to **degaT**

Recommendation

HIGH-01	Admin has an economic incentive to lock user funds forev

Description

Line: DegalSPO.sol#L231

DegalSPO::withdraw allows users to withdraw funds when the contract is not on pause.
DegalSPO::emergencyWithdraw allows users to withdraw funds when the contract is on p
isEmergencyWithdrawEnabled = true. So, the admin can change the owner for **PAUSE_RO**
isEmergencyWithdrawEnabled = false so that users can't withdraw funds. Locked funds s
admin can withdraw rewards by preventing users from withdrawing deposits.
This can be done by using an additional contract that performs calls:

1. **DegalSPO::unpause**
2. **DegalSPO::assignRewards**
3. **DegalSPO::adminWithdraw**
4. **DegalSPO::pause**

As the amount of deposited funds increases, the admin has an economic incentive to mali
and the contract doesn't limit the admin.
In addition, the Polkadot implementation allows users to withdraw DOT without the risk of b
Impact: Admin locks user funds and can use the locked funds to receive rewards.

Recommendation

We recommend avoiding locking user funds. You can add the unlocking of the **DegalSPO::e**
after an arbitrary interval in case of pausing the contract.

HIGH-02	Users can steal funds with negative stETH rebase

Description

Lines: DegalSPO.sol#L213-L214

DegalSPO::withdraw doesn't handle the stETH negative rebase scenario. In this case, the b
unchanged, and the amount of pooled ether decreases.

Thus, the calculation of rewards returns 0 DegalSPO.sol#L327-L331, and the assignment o
accTokenPerShare, **totalSharesDeposited**, **degaTreasuryShares** DegalSPO.sol#L304-L30

Let's look at an example:

```
stETH::getTotalShares = 2 * 10 ^ 18
stETH::getTotalPooledEther = 2 * 10 ^ 18

1) user_1 deposits 10 ^ 18 stETH
user_1.amount = 10 ^ 18
user_1.shares = 10 ^ 18

2) user_2 deposits 10 ^ 18 stETH
user_2.amount = 10 ^ 18
user_2.shares = 10 ^ 18

stETH is negatively rebased by 10%
stETH::getTotalPooledEther = 1.8 * 10 ^ 18

3) user_1 withdraws 9 * 10 ^ 17 stETH
sharesToWithdraw = (9 * 10 ^ 17) * (2 * 10 ^ 18) / (1.8 * 10 ^ 18) = 10 ^ 18
finalWithdrawAmount = (10 ^ 18) * (1.8 * 10 ^ 18) / (2 * 10 ^ 18) = 9 * 10 ^ 17
user_1.amount = 10 ^ 18 - 9 * 10 ^ 17 = 1 * 10 ^ 17
Repeated calculation of shares is incorrect in case of negative rebase
userRemainingShares = (1 * 10 ^ 17) * (2 * 10 ^ 18) / (1.8 * 10 ^ 18) = 1.111... * 10 ^ 17
user_1.shares = 1.111... * 10 ^ 17
```

user_1 can re-withdraw funds and withdraw more funds than necessary. Thus, **user_2** will
funds.

Impact: Users who are the first to withdraw funds can withdraw more than they should. Use

HIGH-03 | Resetting earned rewards upon deposit and partial withdra

Description

Line: **DegalSPO.sol#L184**

Let's say there is a situation where a person makes a repeated deposit after some time. Be
reward-earning efficiency (M-1), this makes sense.

However, during the deposit, all rewards that have already been earned by the user are res
recalculation is performed incorrectly.

Code makes this action:

```
user.debt = (user.amount * accTokenPerShare) / PRECISION_FACTOR;
```

But actually, it should look like this:

```
user.debt += (finalDepositedAmount * accTokenPerShare) / PRECISION_FACTOR;
```

This will allow users to unblock the re-call of the deposit function.

If the user makes a partial withdrawal of funds, then he loses all earned rewards for the per
some of the funds are in staking for the entire period. (Especially if the period between sna

Regardless of the work of the backend, contracts must display relevant and plausible infor
rewards earned over time.

Recommendation

We recommend fixing debt calculation.

MEDIUM-01	The efficiency of receiving rewards decreases for early inv

Description

Early investors who put their **stETH** into the protocol have lower returns over time than late
This is because the profitability of shares transferred to Dega Treasury is no longer consid
It turns out that if the price of tokens after the award is equivalent to the **stETH** spent, then
able to take more tokens than by depositing in your protocol.
Example:
2 users, both have 100 **stETH** (and 100 **shares**) at the start.
User1 deposits 100 **stETH**. (His debt is zero, cause he deposited first)
User2 waits.
Then rebase happens (For clarity, let's take 10%).
Then **assignRewards** function is called.
Let's calculate **assignRewards** and **accTokenPerShare**.

```
currentStAmount ~= getPooledEthByShares(100) ~= 110 tokens
rewardStInt ~= 110 - 100 = 10

accTokenPerShare = 0 + 10 / 100 = 0.1;
totalSharesDeposited ~= 100 - 9.09 = 90.91;

(110 tokens / 100 shares = 10 tokens (profit) / x shares)

degaTreasuryShares ~= 9.09;
```

Then another rebase happens, let's say 10% again:
User2 has now:

```
110 * 1.1 = 121 stETH.
```

Make a call to **assignRewards** function and get:

```
currentStAmount ~= getPooledEthByShares(90.91) ~= 110 stETH

rewardStInt ~= 110 - 100 = 10
```

Client's comments

This behavior is an expected business rule and rewards are distributed by off-chain cod
amount and time period the user has been staking.

MEDIUM-02 | Possible multiplying of **totalStakeTokenDeposited**

Description

Lines: DegalSPO.sol#L152-L154

The **totalStakeTokenDeposited** variable, used as a multiplier at the **deposit()** function, can
compared to the actual value while the divisor's (**poolETHSize**) value remains unchanged.
deposit() (at line DegalSPO.sol#L154) and **withdraw()** (at line DegalSPO.sol#L190) function
Let's consider no one has deposited to the contract. An attacker can sequentially call the **d**
leaving some small amount of **stETH** inside. The rounding errors will lead to the multiple dif
totalStakeTokenDeposited and **poolETHSize** (e.g., after the first such loop, it can be possib
totalStakeTokenDeposited = 2 and **poolETHSize = 1**, which later can be transformed to
totalStakeTokenDeposited = 155984214839625485647458958213110791741060711383
poolETHSize = 1).

Therefore, it can be abused by attackers to block users' funds.

Recommendation

We recommend depositing some dust **stETH** on behalf of some **0xdead** address during the

Client's comments

DEGA will integrate the initial deposit into the deployment execution.

INFORMATIONAL-01	Gas optimization: Custom errors

Description

Lines:

- DegalSPO.sol#L63
- DegalSPO.sol#L87
- DegalSPO.sol#L92
- DegalSPO.sol#L93
- DegalSPO.sol#L131
- DegalSPO.sol#L132
- DegalSPO.sol#L149
- DegalSPO.sol#L169
- DegalSPO.sol#L175
- DegalSPO.sol#L179
- DegalSPO.sol#L186
- DegalSPO.sol#L208
- DegalSPO.sol#L209
- DegalSPO.sol#L231
- DegalSPO.sol#L239
- DegalSPO.sol#L243

Require statements with strings consume more gas and increase bytecode size than custo

Recommendation

We recommend using custom errors

INFORMATIONAL-02	whenNotPaused modifier is redundant for DegalSPO::depo DegalSPO::withdraw

Description

Lines:

- DegalSPO.sol#L168
- DegalSPO.sol#L200

INFORMATIONAL-04	Invalid code style

Description

Events are usually listed before the constructor.

The **MAX_TOTAL_DEPOSIT** is not a constant, so it makes no sense to highlight it in capital

Lines:

- **DegalSPO.sol#L73-80**
- **DegalSPO.sol#L100-104**
- **DegalSPO.sol#L111-116**
- **DegalSPO.sol#L122-128**
- **DegalSPO.sol#L141-146**
- **DegalSPO.sol#L157-166**
- **DegalSPO.sol#L192-198**
- **DegalSPO.sol#L221-228**
- **DegalSPO.sol#L264-268**
- **DegalSPO.sol#L277-281**
- **DegalSPO.sol#L292-296**
- **DegalSPO.sol#L315-318**
- **DegalSPO.sol#L354-356**
- **DegalSPO.sol#L363-365**

Follow NatSpec rules for Solidity and remove @require, @emit, @title statements for functi

here. These statements prevent the code from compiling without errors.

Recommendation

We recommend fixing these issues.

INFORMATIONAL-05	Gas optimizations: memory instead of storag

Description

I. Lines:

DegalSPO.sol#L271

DegalSPO.sol#L284

Lines of code could be optimized, saving variables to memory, or even returning them at o

Instead of:

```
UserInfo storage user = userInfo[_user];

uint256 userRewardBalance = (user.amount * accTokenPerShare) / PRECISION_FACTOR -

return userRewardBalance;
```

Make:

```
UserInfo memory user = userInfo[_user];

return (user.amount * accTokenPerShare) / PRECISION_FACTOR - user.debt;
```

II. Lines:

DegalSPO.sol#L170

DegalSPO.sol#L201

DegalSPO.sol#L232

Working with copies of variables in memory will save a lot of gas; you can edit a memory v then copy it to the storage.

Recommendation

We recommend fixing these issues.

INFORMATIONAL-06	Permit mechanic

Description

INFORMATIONAL-08	Gas optimization: Redundant expressions/varial

Description

Lines:

1. DegalSPO.sol#L25 - the role is unused
2. DegalSPO.sol#L35 - redundant setting to the default value
3. DegalSPO.sol#L41 - the **currentStAmount** variable is unused
4. DegalSPO.sol#L66 - the calculations can be simplified to **10 ** 12**
5. DegalSPO.sol#L92 - the check is needless
6. DegalSPO.sol#L233 - the variable **amountToWithdraw** is unused
7. DegalSPO.sol#L358 - the revert fallback is redundant
8. DegalSPO.sol#L367 - the revert receive is redundant

There are several redundant expressions or variables in your codebase.

Recommendation

We recommend removing/replacing these parts of the code.

INFORMATIONAL-09	Misuse of input amounts instead of final one

Description

Lines:

1. DegalSPO.sol#L176 - **finalDepositedAmount** should be used instead of **_amount**
2. DegalSPO.sol#L217 - **finalWithdrawAmount** should be used instead of **_amount**
3. DegalSPO.sol#L250 - **withdrawnAmount** should be used instead of **amountToWithdr**
   uint256 withdrawnAmount = lidoContract.transferShares(msg.sender, sharesToWit

The provided lines with conditions and emitted events use input or virtual amounts for thes

Recommendation

We recommend using the correct amounts at the provided places.

INFORMATIONAL-10	Gas optimization: Cache storage variables

INFORMATIONAL-11	View function not view in the interface

Description

Line: ILido.sol#L279.

The function **stETH::sharesOf** is **view** in contract **stETH**, but it is not **view** in interface **ILido**. in DegalSPO.sol#L91 will use **CALL** opcode instead of **STATICCALL**.

Recommendation

We recommend changing the function to **view** in the interface.

INFORMATIONAL-12	Redundant variables

Description

Lines:

- DegalSPO.sol#L44 - **debt** variable
- DegalSPO.sol#L39 - **stakedTokenRewardAmount** variable
- DegalSPO.sol#L32 - **accTokenPerShare** variable

The variables listed above are no longer used in the contract.

Recommendation

We recommend the removal of the **debt** from the **UserInfo** struct, **stakedTokenRewardAmo** **accTokenPerShare** variables.

INFORMATIONAL-13	Invalid user.shares calculation

Description

Line: **DegalSPO.sol#L196**

user.shares can have an invalid value, after the next steps:

```
// user1 calls deposit(10): 10 stEth ~ 10 shares

// variables inside Degalspo.sol become:
poolEthSize = 10
totalSharesDeposited = 10
totalStakeTokensDepo = 10

user.shares = 10
user.amount = 10

// +100% rebase
// user1 call withdraw(10)
// internal assignRewards()

rewardStInt = 20 - 10 = 10
sharesToAssignRewards = 5

totalSharesDeposited -= 5 = 10 - 5 = 5
degaTreasury += 5 = 0 + 5 = 5
poolEthSize = 10

// back to withdraw()
userMaxAmount = 10 * 10 / 10 = 10
sharesToWithdraw = getShares(10) = 5
finalWithdrawAmount = getEth(5) = 10
totalSharesDeposited -= 5 = 5 - 5 = 0

user.shares -= 5 = 10 - 5 = 5
```

INFORMATIONAL-14	Unnecessary check

Description

Line: DegalSPO.sol#L217

The check in **emergencyWithdraw()** function is put under the following scenarios:

emergencyWithdraw():

1. Calling **emergencyWithdraw()** with positive rebase and **assignRewards()** called befor
2. Calling **emergencyWithdraw()** with negative rebase and **assignRewards()** called befo

emergencyWithdraw:: scenario 1:

Initial state:

- **user1.amount**: 10 stETH | **totalTokens**: 10 stETH | **totalShares**: 10 shares | **pooledEth**: 10

positive rebase + 10%

Call to **assignRewards()**:

- **totalTokens**: 10 stETH | **totalShares**: 10 - 0.9 = 9.1 shares
- **degaTreasury** = 0.9 shares | **pooledEth**: covertSharesTostTokens(9.1) = 10 stETH

Call to **emergencyWithdraw()**:

- **pooledEth**: covertSharesTostTokens(9.1) = 10 stETH
- **currAmount** = 10 * 10/10 = 10 stETH | **sharesToWithdraw** = 10 * 10/11 = 9.1 shares

Result of the scenario: **sharesToWithdraw(9.1) < user.shares (10)**

---

emergencyWithdraw:: scenario 2:

Initial state:

- **user1.amount**: 10 stETH | **totalTokens**: 10 stETH | **totalShares**: 10 shares | **pooledEth**: 10

negative rebase -50%

Call to **assignRewards()**: there're no rewards

Call to **emergencyWithdraw()**:

- **pooledEth**: covertSharesTostTokens(10) = 5 stETH
- **currAmount** = 10 * 5/10 = 5 stETH | **sharesToWithdraw** = 5 * 10/ 5 = 10 shares

Result of the scenario: **sharesToWithdraw(10) = user.shares (10)**

Based on these scenarios we conclude that the check mentioned in the line is redundant

Recommendation

We recommend removing this check as it does not add any functionality.


STATE
MIND