Three Sigma

Code Audit

Glacier

Glacier A fully automated, liquid staking protocol

Disclaimer

Code Audit

Glacier A fully automated, liquid staking protocol

Disclaimer

The ensuing audit offers no assertions or assurances about the code's security. It cannot be deemed an adequate judgment of the contract's correctness on its own. The authors of this audit present it solely as an informational exercise, reporting the thorough research involved in the secure development of the intended contracts, and make no material claims or guarantees regarding the contract's post-deployment operation. The authors of this report disclaim all liability for all kinds of potential consequences of the contract's deployment or use. Due to the possibility of human error occurring during the code's manual review process, we advise the client team to commission several independent audits in addition to a public bug bounty program.

Table of Contents

Code Audit

Glacier A fully automated, liquid staking protocol

Table of Contents

Summary

Code Audit

Glacier A fully automated, liquid staking protocol

Summary

Three Sigma Labs audited Glacier in a 2 person week engagement. The audit was conducted from 12-07-2023 to 19-07-2023.

Protocol Description

Glacier is a fully automated & truly liquid staking protocol, offering up to 9-11% APY on users' staked AVAX.

It is a liquid staking protocol that gives users direct exposure to the Avalanche network, without the hassle of setting up and managing the systems required to do so. It also doesn't require users to lock their tokens, or have significant upfront capital - meaning they can jump straight in and start earning hourly rewards.

Glacier allows its users to stake any amount of AVAX, and receive staking rewards from the validators on the Avalanche network by simply depositing their funds into a smart contract.

Scope

Code Audit

Glacier A fully automated, liquid staking protocol

Scope

```
contracts/
├── AccessControlManager.sol
├── GlacierAddressBook.sol
├── interfaces
│   ├── IGlacierOracle.sol
│   ├── IglAVAX.sol
│   ├── IGLendingPool.sol
│   ├── IGReservePool.sol
│   ├── IGReserveStrategy.sol
│   └── IWAVAX.sol
└── protocol
    ├── GlacialAVAX
    │   ├── glAVAX.sol
    │   └── wglAVAX.sol
    ├── LendingPool
    │   └── GLendingPool.sol
    └── ReservePool
        ├── GReservePool.sol
        └── strategies
            └── AaveV3Strategy.sol
```

Assumptions

The scope of the audit was carefully defined to include the contracts at the lowest level of the inheritance hierarchy, as these are the ones that will be deployed to the mainnet. The external libraries used in the implementation of these contracts were ones trusted by the community (i.e. OpenZeppelin, Aave) - these libraries have already been battle-tested by multiple protocols, guaranteeing a high level of security.

Methodology

Code Audit

Glacier A fully automated, liquid staking protocol

Methodology

To begin, we reasoned meticulously about the contract's business logic, checking security-critical features to ensure that there were no gaps in the business logic and/or inconsistencies between the aforementioned logic and the implementation. Second, we thoroughly examined the code for known security flaws and attack vectors. Finally, we discussed the most catastrophic situations with the team and reasoned backwards to ensure they are not reachable in any unintentional form.

Taxonomy

In this audit we report our findings using as a guideline Immunefi's vulnerability taxonomy, which can be found at immunefi.com/severity-updated/. The final classification takes into account the severity, according to the previous link, and likelihood of the exploit. The following table summarizes the general expected classification according to severity and likelihood; however, each issue will be evaluated on a case-by-case basis and may not strictly follow it.

Severity / Likelihood	LOW	MEDIUM	HIGH
NONE	None		
LOW	Low		
MEDIUM	Low	Medium	Medium
HIGH	Medium	High	High
CRITICAL	High	Critical	Critical

Project Dashboard

Code Audit

Glacier A fully automated, liquid staking protocol

Project Dashboard

Application Summary

Name	Glacier Pool
Commit	83cb0c3
Language	Solidity
Platform	Avalanche

Engagement Summary

Timeline	12-07-2023 to 19-07-2023
№ of Auditors	2
Review Time	2 person weeks

Vulnerability Summary

Issue Classification	Found	Addressed	Acknowledged
Critical	0	0	0
High	5	4	1
Medium	3	1	2
Low	5	1	4
None	19	1	18

Category Breakdown

Suggestion	9
Documentation	0
Bug	7
Optimization	9
Good Code Practices	7

Code Maturity Evaluation

Code Audit

Glacier A fully automated, liquid staking protocol

Code Maturity Evaluation

Code Maturity Evaluation Guidelines

Category	Evaluation
Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system.
Arithmetic	The proper use of mathematical operations and semantics.
Centralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Code Stability	The extent to which the code was altered during the audit.
Upgradeability	The presence of parameterizations of the system that allow modifications after deployment.
Function Composition	The functions are generally small and have clear purposes.
Front-Running	The system's resistance to front-running attacks.
Monitoring	All operations that change the state of the system emit events, making it simple to monitor the state of the system. These events need to be correctly emitted.
Specification	The presence of comprehensive and readable codebase documentation.
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage.

Code Maturity Evaluation Results

Category	Evaluation
Access Controls	Satisfactory. The codebase has a strong access control mechanism.
Arithmetic	Satisfactory. The codebase uses Solidity version >0.8.0 as well as takes the correct measures in rounding the results of arithmetic operations.
Centralization	Weak. The admin and managers have significant privileges over the protocol.
Code Stability	Satisfactory. The code was stable during the audit.
Upgradeability	Satisfactory. Major contracts can be upgraded by the owner.
Function Composition	Satisfactory. There is little duplicated logic and functions have a clear purpose.
Front-Running	Moderate. There are a few front-running opportunities.
Monitoring	Moderate. Some events are emitted, but not all state changing operations are covered with events.
Specification	Satisfactory. There is a comprehensive and readable documentation.
Testing and Verification	Satisfactory. There is an adequate testing suite with unit, integration, functional and fuzz testing.

Findings

Code Audit

Glacier A fully automated, liquid staking protocol

Findings

3S-GLACIER-H01

AaveV3 RewardsController provides rewards in any token, should be handled separately

Id	3S-GLACIER-H01
Classification	High
Severity	High
Likelihood	High
Category	Bug

Description

The RewardsController from AaveV3 rewards are separate from the yield accrued and can be several different tokens. Thus, they should be handled differently.

Recommendation

Add a separate function in **`AaveV3Strategy`** that allows us to claim the rewards. Returning the rewards is optional, depending on the option chosen below. If a strategy has no extra rewards to be claimed, add the function but only return from it to keep compatibility.

The **`ReservePool`** might end up with tokens other than **`WAVAX`** due to the fact that the rewards distributed might be any token. To address this, there are several options:

1. Add a claim rewards function that sends all rewards to the network wallet, so it can swap for **`wAVAX`** and increase the network balance.

2. Add a swap function in the **`ReservePool`** that swaps all the rewards assets different than **`WAVAX`** for **`WAVAX`**. If the swap fails, send to the **`addresses.networkWalletAddress()`**.

Status

Issue addressed by implementing method 1 in commit: #3a4239f .

3S-GLACIER-H02

`totalReserves` should be fetched from the strategies individually and summed up in the **`ReservePool`**

Id	3S-GLACIER-H02
Classification	High
Severity	High
Likelihood	High
Category	Bug

Description

Currently **`totalReserves`** is updated based on deposits and withdrawal amounts to the **`reservePool`**. This means that rewards accrued in the strategies will not be accounted for, effectively not earning any yield. In the case of AaveV3, the rewards from the RewardsController are separate from the accrued yield from supplying, so only the former would be accounted for.

Recommendation

`totalReserves` should be fetched individually from each strategy and summed up. One way to achieve this is by changing **`totalReserves()`** in **`ReservePool`** to fetch balances from all strategies .

And in the **`AaveV3Strategy`** there should be a function **`getBalance()`** that takes into account the yield accrued.

Status

This issue has been addressed in commit: #4bb9844.

3S-GLACIER-H03

`increaseNetworkTotal()` allows big arbitrage opportunities by depositing before an increase transaction

Id	3S-GLACIER-H03
Classification	High
Severity	High
Likelihood	High
Category	Suggestion

Description

Increasing the network total is done via the **`increaseNetworkTotal()`** function call. Given that **`glAVAX`** allows instant withdrawals, it's very easy for MEV bots to steal rewards from legitimate users. They can front-run an **`increaseNetworkTotal()`** transaction, deposit **`AVAX`**, then the shares/**`wAVAX`** ratio increases and they withdraw, profiting without contributing to the network.

Note that this does not happen in **`rebalance()`** because the ratio is maintained.

Recommendation

Since the network is rebalanced every 24 hours, consider distributing the rewards over a 24h period. This way, bots can't steal yield from the network without having their **`wAVAX`** staked for the next **`rebalance`** call or contributing by letting other users withdraw.

This could be achieved by setting a new state variable **`rewardsPerSecond`** which accrues to **`totalNetworkAVAX`** over a certain time period (24h would be enough because if bots deposited and wanted to claim the full amount they would have to wait 24h, which by then they **`AVAX`** could be used in the network). The bigger the interval, the smaller the MEV window.

Status

This issue has been addressed in commit: #a990358.

3S-GLACIER-H04

`_rebalanceWithdraw()` mechanism in **`glAVAX`** allows arbitrage opportunities by changing the shares/AVAX ratio

Id	3S-GLACIER-H04
Classification	High
Severity	High
Likelihood	High
Category	Suggestion

Description

The balance available for withdrawals is tracked in **`address(this).balance`**. The implementation itself is correct and users will have enough liquidity to withdraw given enough balance is accumulated. However, withdrawing should be done atomically; **`AVAX`** should not be withdrawn when withdraws are rebalanced and the shares burned when the requests are fulfilled.

Essentially, whenever possible changing the ratio shares/**`wAVAX`** ratio should be avoided to prevent MEV opportunities.

Recommendation

Since there are considerable changes involved, a POC was pushed.

In short, instead of withdrawing **`AVAX`** and keeping it in the **`glAVAX`** contract, a reserved **`wAVAX`** amount is tracked, always keeping the shares/**`wAVAX`** ratio intact.

Status

Currently being reviewed by the team.

3S-GLACIER-H05

In **`glAVAX`**, function **`_rebalanceWithdraw()`** withdraws incorrect amount from WAVAX address

Id	3S-GLACIER-H05
Classification	High
Severity	High
Likelihood	High
Category	Bug

Description

In the function **`_rebalanceWithdraw()`** if it is necessary to withdraw from WAVAX in order to satisfy a withdrawal, the code will compare the balance of the WAVAX contract and how much is needed to satisfy the withdrawal. When doing this comparison, if it finds that the balance of the WAVAX contract is bigger than how much is needed to satisfy the withdrawal, then it should only withdraw how much it needs. If the opposite happens it should only withdraw the balance in the WAVAX contract. Currently the code does the opposite.

Recomendation

Change the line in code from:

```
uint256 amountToWithdraw = wavaxBalance > tokenNeeded ?  wavaxBalance : tokenNeeded;
```

To:

```
uint256 amountToWithdraw = wavaxBalance > tokenNeeded ?  tokenNeeded : wavaxBalance;
```

Status

This issue has been addressed in commit: #dad2182.

3S-GLACIER-M01

Strategy withdraw may fail if weights of strategies differ from the real values and might lead to frozen **`ReservePool`**

Id	3S-GLACIER-M01
Classification	Medium
Severity	Medium
Likelihood	Medium
Category	Bug

Description

When depositing or withdrawing in the **`ReservePool`**, it deposits/withdraws individually from the strategies based on the weights. In the case of withdrawals, the transaction might revert.

Suppose default strategy with 50% weight and AaveV3 strategy with 50% weight.

AaveV3 yield reduces, such that its amount is no longer 50%, but 49%.

In the **`ReservePool`**, **`withdraw()`**, the transaction will revert because it tries to withdraw 50% from the AaveV3 strategy, which will revert.

Recommendation

Call the **`getBalance()`** (3S-GLACIER-H02), to get the maximum available balance and withdraw at most this amount. Then, return the actual withdrawn amount, so that **`glAVAX`** can deal with a possibly reduced amount.

Then, in **`glAVAX`**, it has to be dealt with accordingly. The max withdrawal amount check should be done in the **`ReservePool`**, so it is unnecessary to check again in **`glAVAX`**.

Status

Currently being reviewed by the team.

3S-GLACIER-M02

Withdraw snapshot logic can be tricked allowing users to withdraw right away

Id	3S-GLACIER-M02
Classification	Medium
Severity	Medium
Likelihood	Medium
Category	Suggestion

Description

The current withdraw snapshot implementation allows users to withdraw a certain amount after other users withdraw this amount. It is a fair system such that users that want to withdraw a bigger quantity have to wait more time. However, the system can be tricked by splitting the withdrawal amount into smaller parts. For example, someone who wants to withdraw 50000 **AVAX** would have to wait for other people to withdraw 50000 **AVAX** first before they can withdraw. However, if this person placed 1000 withdrawal requests of 50 **AVAX** each, they could withdraw their 50000 **AVAX** just after 50 **AVAX** is withdrawn from other users, not the full 50000.

Recommendation

Withdraw requests are placed whenever there is not enough liquidity to enable user withdrawals. Thus, users have to wait for deposits or `fulfillWithdrawal()` calls from the network. As the current implementation can be tricked, it's safer to fallback to a slightly less elegant solution. Similarly to Benqi Finance, use timestamps to allow users to claim their withdrawal requests. There should be a cooldown period that they have to wait to be able to claim their withdrawal request and a redeem period which after it, the reserved **wAVAX** can be placed again to the network.

Status

Currently being reviewed by the team.

3S-GLACIER-M03

In **`GReservePool`**, if a strategy is frozen reserve pool stops working

Id	3S-GLACIER-M03
Classification	Medium
Severity	High
Likelihood	Low
Category	Bug

Description

If a strategy stops working it will become impossible to use the Reserve Pool. The functions **`deposit()`**, **`withdraw()`** and **`withdrawAll()`** will no longer work. This problem will also affect the contract glAVAX which will make the functions **`rebalance()`** and **`withdraw()`** not work as those functions can call the broken functions of the Reserve Pool.

There is the possibility of the third party application that is being used by the strategy to stop working. Which will leave the funds in the third party application, making you wait for it to recover, and in the meantime the code will revert.

Recommendation

It would be best in this scenario to remove strategies from the reserve pool (keeping them from reverting), this would be possible using the **`rebalance()`** function suggested in 3S-GLACIER-L03 and the fact that applications,like aave, allow for withdrawals if it is frozen.

For this we would add a **`removeBrokenStrategy()`** that would call a function in the strategy to try and withdraw the funds, and would remove the strategy from the list. In the strategy a new function called **`handleBroken()`** would try to withdraw the funds so they can be distrib uted among the other strategies (this function would be tailored to what protocol is being used). A function would also need to be added to return the broken strategy back into the reserve pool.

Status

This issue has been addressed in commit: #56e95e1.

3S-GLACIER-L01

In **`glAVAX`**, should use **`.call`** instead of **`.transfer`**

Id	3S-GLACIER-L01
Classification	Low
Severity	Low
Likelihood	
Category	Suggestion

Description

The functions **`withdraw()`** and **`_claim()`** make transfers directly to the user using **`payable(user).transfer(amount)`**. **`.transfer`** can only forward 2300 gas which means it can fail for some contracts and stop withdraws. **`.call`** should be used instead, for example **`user.call{value: amount}("")`**.

Status

Currently being reviewed by the team.

3S-GLACIER-L02

When changing addresses, use 2 step transfer and/or address 0x0 checks

Id	3S-GLACIER-L02
Classification	Low
Severity	Medium
Likelihood	Low
Category	Suggestion

Description

When changing important addresses, it's best to use additional safety measures to prevent wrong addresses from being set. One layer of protection is using 2 step address transfer, in which a pending role is set first and only then, from the pending role account, it accepts the new role.

Safety measures were added to several of the functions that set addresses (checking if it is a contract). But for example the function `setNetworkWalletAddress()` in `GlacierAddressBook` should have an extra security measure like 2 step transfer (since it can't use the `isContract` check).

Status

Currently being reviewed by the team.

3S-GLACIER-L03

Strategy percentages will differ over time as yield accrued differs in **`ReservePool`**

Id	3S-GLACIER-L03
Classification	Low
Severity	Low
Likelihood	High
Category	Bug

Description

The **`ReservePool`** always deposits and withdraws according to the specified strategy **`weights`**. As the strategies have different yield rates, the actual weights will change over time.

Recommendation

To overcome this, a rebalance function could be added to the **`ReservePool`** that is called whenever the values differ too much or on every rebalance.

Status

This issue has been addressed by adding a rebalance function in commit: #6e95d2a.

3S-GLACIER-L04

`rebalance()` in `glAVAX` reverts if `currentReserves == reserveTarget`

Id	3S-GLACIER-L04
Classification	Low
Severity	Low
Likelihood	
Category	Bug

Description

`rebalance()` in glAVAX deposits to the `ReservePool` if there is available liquidity (`balance`) and the `currentReserves` are smaller than the `reserveTarget`.

The problem is that the `ReservePool` reverts when trying to withdraw 0, which is the case if `currentReserves == reserveTarget.`

Status

Currently being reviewed by the team.

3S-GLACIER-L05

`receive()` in `glAVAX` should only allow `wAVAX`

Id	3S-GLACIER-L05
Classification	Low
Severity	Low
Likelihood	
Category	Suggestion

Description

The current implementation of the `receive()` function in `glAVAX` allows any smart contract to call it, leading to lost funds.

```
receive() external payable {
    // prevents direct sending from a user
    require(msg.sender != tx.origin);
}
```

Recommendation

Refactor the function to only allow transfers from `wAVAX`.

```
receive() external payable {
    require(msg.sender == addresses.wavaxAddress());
}
```

Status

Currently being reviewed by the team.

3S-GLACIER-N01

Usage of **`transferFrom()`** could revert if used from itself

Id	3S-GLACIER-N01
Classification	None
Severity	None
Likelihood	
Category	Suggestion

Description

Under normal circumstances the usage of **`transferFrom()`** in the function **`withdraw()`** in **`AaveV3Strategy`** and in **`GReservePool`** would revert as it would be necessary for the contract to approve itself. When transferring its own funds it should use **`.call()`** instead. **`WAVAX`** does not revert in this instance but other tokens would.

Status

Currently being reviewed by the team.

3S-GLACIER-N02

Strategies in the **`ReservePool`** could be implemented as an array and **`Strategy`** packed

Id	3S-GLACIER-N02
Classification	None
Severity	None
Likelihood	
Category	Optimization

Description

The strategies are stored in the **`ReservePool`** as a mapping **`_strategies`** and length **`_strategyCount`**. This could be reduced to an array of **`Strategy`**, **`Strategy[] public _strategies;`**, increasing readability and gas savings.

Additionally, the struct **`Strategy`** can be reduced to 2 variables and occupy only 1 storage slot by packing the **`logic`** and the **`weight`** together (**`deposited`** is not needed).

Recommendation

Refactor the code to:

```
contract GReservePool is Initializable, IGReservePool, AccessControlManager
{
    ...
    struct Strategy {
        IGReserveStrategy logic; // 20 bytes
        uint96 weight; // so weight can have at most 12 bytes or uint96 to
fill the 32 bytes storage slot
    }
    Strategy[] public _strategies;
    ...
}
```

Status

This issue has been addressed in commit: #4bb9844.

3S-GLACIER-N03

Implement **`_transferShares()`** to prevent having to convert shares/**`wAVAX`** twice in **`glAVAX`**

Id	3S-GLACIER-N03
Classification	None
Severity	None
Likelihood	
Category	Optimization

Description

Sometimes the quantity of shares on hand is available instead of the corresponding **`wAVAX`** amount. In this case, if shares are to be transferred internally, it's easier to use **`_transferShares()`** instead of having the shares, converting to amount and calling **`_transfer()`**, which converts back to shares.

Recommendation

For example, in **`cancel()`**, the shares are available from **`request.shares`**, are converted to **`wAVAX`** amount and then back to shares again in **`_transfer()`**.

Status

Currently being reviewed by the team.

3S-GLACIER-N04

rebalance(), withdrawAmount needed from the **ReservePool** can be simplified

Id	3S-GLACIER-N04
Classification	None
Severity	None
Likelihood	
Category	Optimization

Description

In function **rebalance()**, the **withdrawAmount** from the **ReservePool** can be computed inside the first if statement.

Recommendation

Change the code to

```
function rebalance() external payable isRole(NETWORK_MANAGER) {
    ...
   uint256 withdrawAmount; // new
    if (balance > 0 && currentReserves < reserveTarget) {
        uint256 toFill = reserveTarget - currentReserves;
        uint256 toReserves = toFill > balance ? balance : toFill;
        withdrawAmount = toFill - toReserves; // new
        IGReservePool(reservePoolAddress).deposit(toReserves);
        balance -= toReserves;
    } else {
   ...
}
```

Status

Currently being reviewed by the team.

3S-GLACIER-N05

Checks effects interactions pattern should always be used

Id	3S-GLACIER-N05
Classification	None
Severity	None
Likelihood	
Category	Good Code Practices

Description

The checks-effects-interactions pattern should be used whenever possible, even if apparently it has no consequences. There are some instances specified in the relevant links where it isn't followed.

Recommendation

For example in **`withdraw()`** of **`glAVAX`**

```
...
// Transfer the user with the AVAX
payable(user).transfer(toWithdraw);
_updateWithdrawTotal(toWithdraw);
...
```

Should be instead

```
...
_updateWithdrawTotal(toWithdraw);
// Transfer the user with the AVAX
payable(user).transfer(toWithdraw);

...
```

Status

Currently being reviewed by the team.

3S-GLACIER-N06

If statements can be inverted to increase readability

Id	3S-GLACIER-N06
Classification	None
Severity	None
Likelihood	
Category	Good Code Practices

Description

Inverting if statements can help increase readability and decrease overall code complexity.

Recommendation

Take a look at the following example.

```
function withdraw(uint256 amount) external nonReentrant {
    ...
     if (toWithdraw > 0) {
         <code when toWithdraw > 0>
     }
     emit Withdraw(user, totalWithdrawAvaxAmount);
}
```

It can be replaced by

```
function withdraw(uint256 amount) external nonReentrant {
    ...
     emit Withdraw(user, totalWithdrawAvaxAmount);
     if (toWithdraw <= 0) return;
     <code when toWithdraw > 0>
}
```

Status

Currently being reviewed by the team.

3S-GLACIER-N07

`avaxAmount` is never 0 in **`withdraw()`** in the first **`if (avaxAmount > 0 && ...)`**

Id	3S-GLACIER-N07
Classification	None
Severity	None
Likelihood	
Category	Suggestion

Description

In function **`withdraw()`**, checking **`if (avaxAmount > 0)`** is not required since it is guaranteed to be true by **`require(amount > 0, "ZERO_WITHDRAW");`**

```
function withdraw(uint256 amount) external nonReentrant {
    ...
    require(amount > 0, "ZERO_WITHDRAW");
    ...
    if (avaxAmount > 0 && depositAmount > 0) { // avaxAmount is never 0
       ...
    }
    ...
```

Recommendation

Remove **`avaxAmount > 0`** from the if statement.

Status

Currently being reviewed by the team.

3S-GLACIER-N08

Withdraw requests could be stored in a simpler way in **glAVAX**

Id	3S-GLACIER-N08
Classification	None
Severity	None
Likelihood	
Category	Optimization

Description

Global and user withdrawal requests are stored in a mapping and a variable tracking the length respectively. Thus, it's easier to use 1 array of global withdraw requests and a mapping with an array for each user with their individual withdraw requests, pointing to the global array of withdraw requests. The functions that would be affected by this change are **_withdrawRequest()**, **_claim()**, **_cancel()**, **claim(), claimAll()**, **cancelAll()** and **cancel()**.

Recommendation

When creating a new withdraw request in **_withdrawRequest()**, do

```
uint256 globalWithdrawRequestId = withdrawRequests.push(request) - 1;
userWithdrawRequests.push(globalWithdrawRequestId);
```

When claiming or cancelling individually, do

```
uint256 globalId = withdrawRequests[user][id]
uint256 lastUserRequest = withdrawRequests[user].length - 1;
if (lastUserRequest != 0) withdrawRequests[user][id] =
withdrawRequests[user][lastUserRequest];
withdrawRequests[user].pop();
withdrawRequest = withdrawRequests[globalId];
...
delete withdrawRequests[globalId];
```

Note that on **`claimAll()`** and **`cancelAll()`**, looping through the method above could cause issues because it changes the position of the current index with the last index in each iteration. In that case, it would make more sense to

1. cache the length of the array
2. loop through the array, starting at the last index
3. pop the element at the end of the body of the loop

Status

Currently being reviewed by the team.

3S-GLACIER-N09

In **glAVAX**, checking that **amount > 0** earlier can save some gas

Id	3S-GLACIER-N09
Classification	None
Severity	None
Likelihood	
Category	Optimization

Description

In the function **withdraw()** the function **_borrowLiquidity()** is called to borrow funds from the lending pool if necessary. **_borrowLiquidity()** eventually calls **_lendAvax()** that checks the following conditions **if (amount > 0 && lendingBalance > 0)**. The condition **amount > 0** could be checked instead in the **withdraw()** function before calling **_borrowLiquity()**, this would make it so that when **amount == 0** no gas is spent going into the function **_borrowLiquidity()**.

Recommendation

Put check before calling **_borrowLiquidity()**:

```
if( avaxAmount > 0) {
      avaxAmount = _borrowLiquidity(avaxAmount);
}
```

And remove check in **_lendAvax()**:

```
if (lendingBalance > 0) {
```

Status

Currently being reviewed by the team.

3S-GLACIER-N10

In **glAVAX**, naming convention for storage variable should be consistent

Id	3S-GLACIER-N10
Classification	None
Severity	None
Likelihood	
Category	Good Code Practices

Description

The code follows the convention that storage variables start with an _. In **glAVAX** some variables have the _ others do not. They should all follow the same rules to increase code readability.

Status

Currently being reviewed by the team.

3S-GLACIER-N11

Storage variables should be cached whenever possible to save gas

Id	3S-GLACIER-N11
Classification	None
Severity	None
Likelihood	
Category	Optimization

Description

Storage reads should be avoided whenever possible to save gas, which can be achieved by caching the variables in memory.

Recommendation

For example, in **`_withdrawRequest()`**, the variables **`totalWithdrawRequests`** and **`userWithdrawRequestCount`** can be cached.

Status

Currently being reviewed by the team.

3S-GLACIER-N12

Unnecessary user balance check in **`_withdrawRequest()`**

Id	3S-GLACIER-N12
Classification	None
Severity	None
Likelihood	
Category	Optimization

Description

The balance of the user when withdrawing against the amount passed in as argument is already checked in the **`withdraw()`** function, there's no need to check again in **`_withdrawRequest()`**.

Recommendation

Remove the following line

```
function _withdrawRequest(uint256 amount) internal {
    address user = msg.sender;
    require(amount <= balanceOf(user), "INSUFFICIENT_BALANCE"); // remove
this line
...
}
```

Status

Currently being reviewed by the team.

3S-GLACIER-N13

Typo in **`fufillWithdrawal()`** in **`glAVAX`**, should be **`fulfillWithdrawal`**

Id	3S-GLACIER-N13
Classification	None
Severity	None
Likelihood	
Category	Good Code Practices

Description

Typo in **fufillWithdrawal()**.

Status

Currently being reviewed by the team.

3S-GLACIER-N14

In **`GReservePool`**, remove unnecessary logic

Id	3S-GLACIER-N14
Classification	None
Severity	None
Likelihood	
Category	Optimization

Description

In the function **`withdraw()`**, the variable **`amountTransfered`** is going to be equal to the biggest of two variables, either **`totalWithdraw`** or **`amount`**. Since **`totalWithdraw`** will either be bigger than **`amount`** or the same, there is no need to check if one is bigger than the other as **`totalWithdraw`** will always be the correct value. Therefore you can just use **`totalWithdraw`** for the transfer.

Status

Currently being reviewed by the team.

3S-GLACIER-N15

In **`GReservePool`**, use constants naming conventions

Id	3S-GLACIER-N15
Classification	None
Severity	None
Likelihood	
Category	Good Code Practices

Description

There are two constants in this contract (**`_maxStrategies`** and **`_defaultStrategyWeigth`**), they should use the naming convention of being in capital letter the same way it is done in the other contracts. They should be **`MAX_STRATEGIES`** and **`DEFAULT_STRATEGY_WEIGHT`**.

Status

Currently being reviewed by the team.

3S-GLACIER-N16

In **`GLendingPool`**, remove incorrect comment

Id	3S-GLACIER-N16
Classification	None
Severity	None
Likelihood	
Category	Suggestion

Description

In the function **`repay()`** the NatSpec comment says **`Anyone can repay a loan for anyone`**. This was true in the older version but not anymore, therefore this part of the comment should be removed.

Status

Currently being reviewed by the team.

3S-GLACIER-N17

In **`GLendingPool`**, event parameters can be indexed

Id	3S-GLACIER-N17
Classification	None
Severity	None
Likelihood	
Category	Good Code Practices

Description

In **`GLendingPool`** the events **`Borrowed`** and **`Repaid`** can have the parameter **`user`** as **`indexed`**.

Status

Currently being reviewed by the team.

3S-GLACIER-N18

Remove unused imports

Id	3S-GLACIER-N18
Classification	None
Severity	None
Likelihood	
Category	Optimization

Description

If an import is never used it should be removed to save on code size.

In **`wglAVAX`** imports **`PausableUpgradeable`**, **`IWAVAX`**, **`IGReservePool`**, **`IGLendingPool`**, **`AcessControlManager`** and **`GlacierAddressBook`** are never used.

In **`GLendingPool`** import **`console`** is not used.

Status

Currently being reviewed by the team.

3S-GLACIER-N19

NatSpect comments should include explanations of parameters and return variables

Id	3S-GLACIER-N19
Classification	None
Severity	None
Likelihood	
Category	Good Code Practices

Description

The NatSpect comments of most of the functions in the contracts only have an explanation to what the function does. It is good practice to also add an explanation of the inputs and output of the functions.

Status

Currently being reviewed by the team.