PUBLIC

Code Assessment

of the Summer Earn Protocol Smart Contracts

January 14, 2025

Produced for

summer.fi

by

CHAINSECURITY

Contents

1 Executive Summary

Dear Summer.fi Team,

Thank you for trusting us to help Summer.fi with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of Summer Earn Protocol according to Scope to support you in forming an opinion on their security risks.

Summer.fi primarily implements an investment protocol, where users can deposit their funds in a so called "fleet" which is bound to an underlying asset. The liquidity of each fleet is then managed by some trusted manager. The whole protocol is globally managed by a DAO controlled by the Summer token. Summer.fi also implements dutch auctions and rewards distribution for Summer Earn, an ark for Pendle PT token, a contract to batch user interactions with the system, and a contract to manage the vesting of Summer token.

The most critical subjects covered in our audit are asset solvency, internal accounting, functional correctness, and access control. Security regarding all the aforementioned subjects is good. Please note that several issues have been marked as risk accepted.

Security regarding internal accounting has been improved after fixing the issue State Not Updated Before Staking. Security regarding asset solvency has been improved after fixing the issue Disembarking AaveV3Ark Can Fail. Security regarding functional correctness has been improved after fixing the issues Tip Not Collected and Wrong Order Assumption in Withdrawable Arks Caching. The most critical issues have been addressed after the first intermediate report but some issues were introduced with the fixes, see Wrong Direction for Buffer Adjustment Checks.

The general subjects covered are gas efficiency, trustworthiness, specification, and code complexity. Security and quality regarding all the aforementioned subjects is high. Gas efficiency is good but can be enhanced further, see Gas Optimizations. See Power of AdmiralsQuarters role, Power of Governance and How to choose auction parameters for findings regarding privileged actions. See BuyTokens Can Revert From Frontrunning and Sequencer Downtime Can Influence Auction Price for findings regarding MEV. See Misnaming of Quadratic Decay Function for findings on terminology.

We want to highlight the assumptions over the governor role, i.e. the governor role is unique and is controlled only by the DAO, and its power over the system, see Roles and Trust Model and Power of Governance. We also want to highlight the limited use cases of `PendlePtOracleArk`, see System Overview and Excluded from scope.

In summary, we find that the codebase provides a good level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered, and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	5
• Code Corrected	5
Medium-Severity Findings	13
• Code Corrected	10
• Risk Accepted	3
Low-Severity Findings	28
• Code Corrected	22
• Specification Changed	2
• Code Partially Corrected	1
• Risk Accepted	1
• Acknowledged	2

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the Summer Earn Protocol repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	18 Nov 2024	44249522ab2b14a659d5e671b0975fd80ef1a77a	Scope 1 - Initial Version
2	02 Dec 2024	2650345771c8dcd1287eadfff1789f6010cb4351	Scope 2 - Initial Version
3	02 Dec 2024	2650345771c8dcd1287eadfff1789f6010cb4351	Scope 3 - Initial Version
4	11 Dec 2024	4d8695c9e91e34830c5700d83c22b0f33a4479af	Scope 1 - First fixes
5	19 Dec 2024	8ca82a020b48b6cb2aaa0c67f2a27723903a2df2	Scope 2 - First fixes
6	06 Jan 2025	db0c88213ec745b2976debeae06e0035f4c91328	Scope 1 - Second fixes
7	06 Jan 2025	db0c88213ec745b2976debeae06e0035f4c91328	Scope 3 - First fixes
8	07 Jan 2025	8278334b5b0b8aba331fb2dc18ef6f5b162fe2e2	Scope 2 - Second fixes
9	14 Jan 2025	c64bcad3081377c64ac5663e109406b50c6f2032	Scope 3 - Second fixes and final version for all scopes

For the solidity smart contracts, the compiler version `0.8.28` was chosen.

The scope of the review is separated into the following subscopes:

Scope 1:

```
packages/access-contracts/src/contracts/
    LimitedAccessControl.sol
    ProtocolAccessManaged.sol
    ProtocolAccessManager.sol
packages/core-contracts/src/contracts/
    arks/
        AaveV3Ark.sol
        BufferArk.sol
    libraries/
        StorageSlots.sol
    Ark.sol
```

```
    ArkAccessManaged.sol
    ArkConfigProvider.sol
    AuctionManagerBase.sol
    ConfigurationManaged.sol
    FleetCommander.sol
    FleetCommanderCache.sol
    FleetCommanderConfigProvider.sol
    FleetCommanderPausable.sol
    Raft.sol
    TipJar.sol
    Tipper.sol
packages/core-contracts/src/utils/CooldownEnforcer/
    CooldownEnforcer.sol
```

Scope 2:

```
packages/dutch-auction/src/
    DecayFunctions.sol
    DutchAuctionLibrary.sol
    DutchAuctionMath.sol
packages/core-contracts/src/contracts/
    FleetCommanderRewardsManager.sol
packages/rewards-contracts/src/contracts/
    StakingRewardsManagerBase.sol
```

Scope 3:

```
packages/core-contracts/src/contracts/
    arks/
        PendlePtOracleArk.sol
    AdmiralsQuarters.sol
packages/core-contracts/src/utils/exchangeRateProvider/
    CurveExchangeRateProvider.sol
    ExchangeRateProviderBase.sol
packages/gov-contracts/src/contracts/
    SummerVestingWallet.sol
```

2.1.1 Excluded from scope

All the files and contracts not explicitly listed are out of the scope of this review. Third-party contracts and libraries are out of the scope of this review and are assumed to work as intended. In particular, the PRBMath external library is out-of-scope. Integrated protocol, in particular Pendle, protocols used when swapping on Pendle, and Curve, are out of the scope of this review and are assumed to work as intended. The `PendlePtOracleArk` is assumed to be only used with a `USDe` market on Arbitrum One (which is not yet available), and the Curve pool is assumed to be the following: https://arbiscan.io/address/0x1c34204fcfe5314dcf53be2671c02c35db58b4e3.

3 System Overview

This system overview describes the initially received version (Version 1) of the contracts as defined in the Assessment Overview.

At the end of this report section, we have added subsections for each of the changes according to the versions.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

3.1 Scope 1

This scope focuses on the core of Summer Earn Protocol. Summer.fi offers an investment protocol, where users can deposit their funds in a so called "fleet" which is bound to an underlying asset. Each fleet is led by a `FleetCommander` contract, which is the main entry point for users and liquidity management. The keeper of a fleet is responsible for distributing the funds across multiple third-party protocols to maximize the yield. The protocol initially holds users' funds in a buffer to facilitate withdrawals. Furthermore, accrued rewards from the third-party protocols are sold in a Dutch Auction. The tokens used to buy these reward tokens are later reinvested into the protocol to accrue even further yield and rewards. Keepers of the protocol can rebalance the funds distribution among third-party protocols.

In what follows, we delve into different components of the system.

3.1.1 Access management system

Access management is accomplished through a `ProtocolAccessManaged` contract. It queries a `ProtocolAccessManager` contract to fetch different roles in the system and through it enforce access control. System roles can be divided into two categories:

1. Static roles: These roles are valid globally e.g., `GOVERNOR_ROLE`, `SUPER_KEEPER_ROLE`, `GUARDIAN_ROLE`, and `DECAY_CONTROLLER_ROLE`.
2. Contract specific roles: These roles are authorized within a contract e.g., `CURATOR_ROLE`, `KEEPER_ROLE` and `COMMANDER_ROLE`.

`GOVERNOR_ROLE` is granted upon deployment of access management system and can grant or revoke a set of well-specified static as well as contract specific roles during runtime. If an entity gets compromised, it can renounce its roles.

3.1.2 Address oracle system

`ConfigurationManaged` keeps the address of the `ConfigurationManager` contract, which can be queried to get some system-wide addresses e.g., `treasury`. `GOVERNOR_ROLE` has the authority to modify these addresses.

3.1.3 Arks

Arks in the system can be categorized to two different types:

1. External Ark: when an `Ark` is a wrapper for a third-party protocol. It should provide the following main functionalities:

- `sweep()` to move arbitrary tokens to the `Raft` (covered later) or to move the fleet's underlying asset to the buffer ark.
- `board()` to receive the fleet's underlying tokens and deposit them into the underlying protocol.

- `disembark()` to withdraw from the underlying protocol and send the tokens back to the caller.
- `harvest()` to get the accrued rewards from the underlying protocol.
- `move()` to withdraw from the underlying protocol and approve it for another ark and finally board it into the destination ark.

Depending on the API of the underlying protocol the actual implementation of the aforementioned functions can be different.

2. Buffer Ark: is unique for each fleet commander. All users' deposits and withdrawals go through this entity. Which means that when users deposit in the system, their tokens are initially moved to the buffer ark. The keeper will later decide to move them to other arks in order to actually invest in the third-party protocols.

3.1.4 TipJar

Governance uses this contract to define the distribution of rewards from the active fleet commanders among the receivers. Governance can:

- Add tip stream: a tip stream is valid and can be added if:
    1. the recipient is not already registered
    2. total allocation plus this allocation does not go beyond 100%.
- Remove tip stream: removal of a tip stream is possible if its locking period has passed.
- Update tip stream: an existing tip stream can be updated after its locking period has passed if total allocation does not go beyond 100%. Upon updating a tip stream, governance can decide to **shake** all the active fleet commanders:

By shaking a fleet commander, TipJar's shares of the fleet commander (explained in the next subsection) are redeemed to receive the underlying tokens in return. The received tokens are then distributed with respect to the allocation of tip streams among the receivers. If the total allocation is less than 100%, the remaining amount gets transferred to the system's treasury.

Note that shaking a fleet commander can be done without governance intervention. However, shaking of a specific fleet commander is possible only if it is active.

3.1.5 FleetCommander

This smart contract extends the functionality of an ERC4626 tokenized vault; it mediates the users' interaction with the system (depositing, minting, redeeming, and withdrawing) as well as distributing rewards. When a fleet commander gets deployed, one buffer ark gets automatically deployed for it as well. This buffer ark is marked as withdrawable from the beginning.

The `FleetCommander` contract exposes the following functionalities:

1. Accruing Tips: FleetCommander inherits the Tipper contract. It, exponentially with time, mints new shares:

$$shareIncrease = currentSupply * (1 + \frac{tipRate}{SECONDS\_PER\_YEAR})^{(currentTime - lastTipTime)} - currentSupply$$

and increases the previous timestamp to the current time. The newly minted shares are sent to the `TipJar` and as explained previously upon shaking the fleet commander, these shares get redeemed and distributed among the tip stream receivers. It is important to notice that users are incentivized to participate in the system if the tip rate and its growth are smaller than the weighted yields of the integrated protocols and their growth.

All the operations of depositing or withdrawing from the fleet commander are preceded by accruing the tips and we avoid mentioning them for each occurrence.

2. Depositing/Minting: Users can deposit their assets (in form of the fleet underlying token) and receive LP shares. The amount of assets a user can deposit is capped by the deposit cap of the fleet commander. When depositing, the payment tokens of the user are first transferred to the fleet commander, then the user receives their LP tokens, and finally the payment tokens get boarded into the buffer ark.
3. Adjusting the buffer: Keepers can move funds from other arks to the buffer ark or vice versa. If funds are moved from the ark, its balance should not drop below the configured minimum balance. Through this operation, users' funds boarded in the buffer ark (upon depositing/minting) get boarded to the third-party protocols.
4. Rebalancing: Keepers can perform rebalancing operations. When rebalancing, funds are moved from a source ark to a destination ark through the following steps:
    - assuring that both source and destination arks are active (destination's deposit cap is not zero), and that in/outflow of the funds is below the defined thresholds.
    - the buffer ark is not involved in the rebalancing (rebalancing the ark buffer is done by adjusting it).
    - disembarking from the source arks and boarding to the destination ark.

Rebalancing and adjusting the buffer ark cannot be done more frequently than once per cooldown time. However, governance has the privilege to forcefully perform them. Rebalancing is also the only way to disembark arks that are not marked as withdrawable.

5. Withdrawing/Redeeming: Withdrawal's call path depends on whether there are enough underlying asset tokens present in the buffer Ark or not, given that the user possesses enough shares of the vault. If the buffer ark has enough assets to cover this Withdrawal, the user's shares are burned, and the respective amount of assets is transferred to the user. However, if what the user wants and is allowed to withdraw is more than the buffer's assets, it iterates over the arks sorted according to the value of their managed assets in an ascending order and tries to pay the assets back to the user starting with the arks with the lowest amount of assets. An ark is considered to be withdrawable if it does not require the keeper data. Otherwise, it would be skipped from the withdrawal list.

Governance can extend the set of the arks orchestrated by a fleet commander. Each ark can be added to only one fleet commander. If it does not require the keeper data, it gets marked as withdrawable and would later be used when withdrawing from the system. Similarly, governance can remove an ark from a fleet commander if the ark is active and does not hold any assets.

3.1.6 Raft

This contract manages auctions for arks orchestrated by its associated fleet commander. The governance can sell ark tokens either by

1. sweeping the arbitrary ERC20 tokens in the ark
2. or by collecting the rewards of the underlying protocol

and sells them in a Dutch auction for some payment tokens. As the governance starts the auction, it receives a portion of auction tokens as kicker reward. Note that the governance should be able to manage its balance for arbitrary rewards token.

Users willing to buy these ERC20 tokens can call the `buyTokens` function on the `Raft`, which will transfer the according amount of payment tokens to the `Raft` contract. When the end time of the auction comes, or when there are no more tokens to be sold in the auction, the auction gets settled. If the underlying ark does not require keeper data, the Raft boards the received payment tokens back into the ark.

3.1.7 Changes in Version 2

- a new role `ADMIRALS_QUARTERS_ROLE` is defined in the `ProtocolAccessManager`
- bearers of the `GUARDIAN_ROLE` can have an expiration, this can be checked with the `isActiveGuardian()` function from `ProtocolAccessManager`. After expiry, they will still have the role in the system, the expiry is enforced only through `isActiveGuardian()`.
- the `FleetCommanderConfigProvider` does not store whether an ark is withdrawable or not anymore, withdrawability is checked directly from the ark through the new `withdrawableTotalAssets()` function
- the rebalance in- and outflows of arks are enforced over the whole rebalancing operations batch. The limits for the flows are enforced over the absolute cumulative values for each direction individually.
- sweeping and starting auctions from the `Raft` are permissionless. Sweepable tokens are whitelisted, the set of sweepable tokens per ark is set by each ark's `CURATOR_ROLE` bearer. Each `(ark, rewardToken)` has dedicated auction parameters, set by the ark's `CURATOR_ROLE` bearer.
- harvesting tokens on arks from the `Raft` is permissioned
- the tipping formula has been updated to :

$$shareIncrease = (currentSupply - TipJarBalance) * \frac{(currentTime - lastTipTime) * tipRate}{SECONDS\_PER\_YEAR}$$

3.1.8 Changes in Version 3

- only the registered commander, raft, or another active ark can board liquidity into an ark, not any address with a commander role anymore
- the `onlyCommander` modifier has been updated to allow only the registered commander

3.2 Scope 2

This scope focuses on the Dutch Auction Library and the FleetCommanderRewardsManager, which are used in the SummerFi Earn protocol.

3.2.1 Dutch Auction Library

The DutchAuctionLibrary implements logic for dutch auctions.

A dutch auction starts at the `startPrice` and reduces the price over time, down to the `endPrice`. The auction ends when the `endPrice` is reached or when all tokens have been sold. Users can always buy tokens at the current price. There are two price decay functions implemented: a linear decay function and a quadratic decay function.

The contract exposes the following functions:

- `createAuction()` starts a new auction, given the auction parameters. A `kickerReward` is given to the caller. The tokens to be auctioned should already be present in the calling contract. The calling contract must implement access control or default values for the auction parameters. Allowing arbitrary values for the auction parameters could lead to draining the contract, see How to choose auction parameters
- `getCurrentPrice()` computes the price for a given auction at the current point in time. It uses `block.timestamp`. The price decreases over time following the decay function selected at the start of the auction.

- `buyTokens()` allows any user to buy some of the auctioned tokens at the current price. The user can specify an amount of tokens to buy. The contract takes the corresponding amount in payment tokens and gives the user auction tokens. If there are still tokens left to sell, the auction stays open, otherwise it is finalized.
- `finalizeAuction()` must be called after the auction is over. It will transfer any remaining tokens to the `unsoldTokensRecipient`.

Note that these are internal functions acting on data in storage and memory that the derived contract must call explicitly.

3.2.2 FleetCommanderRewardsManager

The FleetCommanderRewardsManager allows users to stake their FleetCommander tokens and earn additional rewards. The rewards can be multiple different tokens. Reward amounts are decided by the `governor` role. The tokens to be given as rewards must be present in the contract. The rewards are distributed at a constant rate every second, proportionally to the amount of FleetCommander tokens staked.

It exposes the following state-mutating functions:

- `stake()` allows a user to deposit any amount of FleetCommander tokens into the contract and start earning rewards.
- `unstake()` allows a user to take back any amount of FleetCommander tokens. They will no longer be earning rewards on these tokens.
- `exit()` allows a user to take back all of their staked FleetCommander and claim their rewards in one call.
- `stakeOnBehalfOf()` allows anyone to stake FleetCommander tokens and give the resulting stake to any receiver.
- `unstakeOnBehalfOf()` allows anyone to unstake their own FleetCommander tokens and give the unstaked tokens to any receiver. It also allows the AdmiralsQuarters contract to unstake any user's tokens and direct the FleetCommander tokens to an arbitrary address (the current implementation of AdmiralsQuarters unstakes the tokens to itself).
- `getReward()` allows a user to claim their rewards in all of the distributed tokens.
- `notifyRewardAmount()` allows the governor to start distributing rewards. The tokens must already be in the balance of the contract. If there is already a reward campaign running for the same token, it is overriden but the duration of the new campaign must be the same. If the token has never been used as a reward token before, it is added to the reward token list.
- `setRewardsDuration()` allows the governor to change the reward duration for a given token. This is not allowed if a campaign is still ongoing.
- `removeRewardToken()` allows the governor to remove a reward token from the contract and delete the data in storage. This is not allowed if a campaign is still ongoing.

3.2.3 Changes in Version 2

- In FleetCommanderRewardsManager, the `unstakeOnBehalfOf` function has been replaced with `unstakeAndWithdrawOnBehalfOf`. It takes a new parameter `claimRewards`. When it is set to `true`, the function claims the user's staking rewards in addition to unstaking for them.
- The FleetCommanderRewardsManager can no longer unstake funds to an arbitrary address. It must always send the unstaked funds to the user address. This limits the trust requirements of the AdmiralsQuartersRole. It can no longer drain all funds.

- In StakingRewardsManagerBase, `rewardRate` is now internally stored scaled by WAD.

3.3 Scope 3

This scope focuses on a new ark and two non-core contracts, and can be split into three subscopes, where Summer.fi offers:

1. `PendlePtOracleArk`: an ark to integrate with Pendle market by swapping the fleet's asset into the market's Principal Token (PT)
2. `AdmiralsQuarters`: a contract to batch user's interactions with Summer Earn Protocol
3. `SummerVestingWallet`: a vesting contract for the distribution of the Summer token with two different vesting schedules

3.3.1 `PendlePtOracleArk`

This ark allows to enter a Pendle market by swapping the fleet's underlying asset into the market's PT token. In Pendle, a market has an expiration date, when reached, the ark must "rollover" the liquidity to a new market, set by governance, in order to continue collecting the yield. In addition to its `asset` (fleet's underlying asset), the ark defines a `marketAsset`, which is the Pendle market's denomination asset.

- rollover: rollover can be triggered by arbitrary addresses, or automatically when boarding/disembarking liquidity. For a rollover to take place, the current market must be expired, the new market must have been set. If not, rollover will revert. Furthermore, the new market's oracle for Pendle LP pricing must be ready. If any of these conditions is not met, the rollover call will be a no-op. Then, the rollover happens in three distinct steps:

    1. the whole PT balance is first redeemed for the market's Standardized Yield (SY), and then the SY are redeemed for `marketAsset` on the current market.
    2. the market is updated on the ark. The SY, PT, and Yield Token (YT) are overwritten with the tokens of the new market, the `marketAsset` is checked to be an input and an output token on the new SY, and the new market's expiration date is recorded.
    3. swap the `marketAsset` into the new market's PT with slippage protection with Pendle's `swapExactTokenForPt()` action. The minimum amount out for slippage protection is determined as follows:

    $$\frac{amount_{marketAsset}}{PtToAssetRate_{PendleTWAPOracle}} * (1 - slippagePercentage)$$

- boarding: performs rollover if needed. Boarding can only be done if the market expiration date is in more than 20 days and the rate for `asset/marketAsset` is within some bounds in the corresponding Curve pool. Swaps the boarded ark's `asset` into the market's PT with Pendle's `swapExactTokenForPt()` action. The `asset` will first be swapped into `marketAsset` before being used in Pendle. The keeper is trusted to set correct parameters for the swaps.
- disembarking: checks and does rollover if needed. Disembarking can only be done if the rate for `asset/marketAsset` is within some bounds in the corresponding Curve pool. If the rollover call was a no-op, and the current market is expired, withdrawing from Pendle is also a no-op and disembarking is likely to fail, as the `amount` will not be transferrable from the ark. In the case where disembarking can be done, some of the ark's PT will be swapped for the amount of `asset` to be disembarked with Pendle's `swapExactPtForToken()` action. The PTs are first redeemed on Pendle to some allowed output tokens on the market and then swapped into `asset`. The keeper is trusted to set correct parameters for the swaps.
- withdrawal by governor: if the market is expired, the governance (`GOVERNOR_ROLE`) can redeem the ark's PT for `marketAsset` in the same way as in point 1. of rollover, and transfer the `marketAsset` to itself.

This ark is intended only for use with Pendle's USDe market on Arbitrum One as the ark requires an exchange rate of `1:1:1` for `marketAsset:SY:PT` after market expiration to work properly.

3.3.2 AdmiralsQuarters

This contract is intended to be the entry point of Summer Earn Protocol for users, it allows to bundle actions with the use of `multicall`. Users can do the following actions through the `multicall` function:

- deposit tokens to the contract
- withdraw tokens from the contract
- withdraw from their CompoundV3 position to the contract. Prior to this, the user must set the `AdmiralsQuarters` as an allowed operator on CompoundV3.
- withdraw their AaveV3 position to the contract. Prior to this, the user must approve the `aToken` to the `AdmiralsQuarters`.
- redeem their shares of an `ERC4626` vault to the contract. Prior to this, the user must approve the shares to the `AdmiralsQuarters`.
- swap tokens already present in the `AdmiralsQuarters` through the 1inch router
- deposit tokens present in the `AdmiralsQuarters` to an active `FleetCommander`. The shares are minted to some `receiver` address set by the user
- withdraw tokens from an active fleet. The withdrawn tokens are sent to the `AdmiralsQuarters`, the user must include a token withdrawal multicall action to exit their tokens.
- stake their shares of a `FleetCommander` into the `StakingRewardsManager` linked to that `FleetCommander`
- unstake from a `StakingRewardsManager` and withdraw their shares from the corresponding `FleetCommander`. The withdrawn tokens are sent directly to the user

The owner of the contract can rescue arbitrary tokens from the `AdmiralsQuarters`.

3.3.3 SummerVestingWallet

This vesting wallet extends the OpenZeppelin's `VestingWallet` by adding some protocol-specific access management, especially a guardian role, granted to some arbitrary guardian address passed as constructor parameter. It also redefines the vesting schedule with two separate modes:

- team vesting: after a cliff period of 180 days, releases the tokens after each new quarter over 2 years. On top of this, an address with the `GUARDIAN_ROLE` can set some goals with some token amounts bound to the goals. An with the `GUARDIAN_ROLE` can the mark a goal as reached, this will immediately release the token bound to that goal. The amounts related to the goals are not subject to the vesting schedule.
- investor or ex-team vesting: after a cliff period of 180 days, releases the tokens after each new quarter over 2 years.

An address with the `GUARDIAN_ROLE` can withdraw the amount of tokens bound to a goal that is not yet reached. The contract is expected to be used to distribute one token only, the Summer token. The contract is likely to not work for distributing tokens other than the main distributed token.

3.3.4 Changes in Version 2

- the `AdmiralsQuarters` has been updated to allow users to claim their rewards at the same time they unstake and withdraw their shares from a `StakingRewardsManager`. This is done by adding a new function `unstakeAndWithdrawOnBehalfOf()` that calls the `_unstake()` and optionally `_getReward()` functions in the `StakingRewardsManager` in a single transaction.

3.4 Roles and Trust Model

Scope 1

- Users are fully untrusted
- Bearer of the `GOVERNOR_ROLE` is expected to be unique and be granted to a timelock contract managed by a DAO (the governance), itself controlled by the Summer token holders. The governance is generally trusted to act non-maliciously, but it is important to highlight that it can be attacked or misused, see Power of Governance. This role is the most powerful of the system as it can grant and revoke all the other roles, as well as taking critical actions in the protocol.
- Bearers of the `SUPER_KEEPER_ROLE` are expected to always act in the best interest of the protocol. In particular, since they can act on every fleet, they are trusted to rebalance fleets' arks and adjust buffers in order to maximize the return.
- Bearers of the `GUARDIAN_ROLE` are expected to always act in the best interest of the protocol. In particular, they are trusted to pause and unpause `FleetCommanders`, and pause the `TipJar` when necessary.
- Bearers of the `DECAY_CONTROLLER_ROLE`. No contract makes use of this role in the scope of this review. No assumption is made for this role.
- Bearers of the `CURATOR_ROLE` for a particular `FleetCommander` are expected to always act in the best interest of the protocol. In particular, as they are responsible for assessing and managing risks, they are trusted to set the risk management parameters of the fleet with the goal of maximizing the yield and users' fund risk exposure in the different integrated protocols.
- Bearers of the `KEEPER_ROLE` for a particular `FleetCommander` are expected to always act in the best interest of the protocol. In particular, they are trusted to rebalance their fleet's arks and adjust the buffer in order to maximize the return.
- Bearer of the `COMMANDER_ROLE` for a particular `Ark` is expected to be unique per ark, it is also expected to be a `FleetCommander` with a matching underlying asset.

Scope 2

In Version 1, in the FleetCommanderRewardsManager contract, the AdmiralsQuartersRole had to be given to a contract that returns any funds it unstakes using `unstakeOnBehalfOf` to the user it unstaked for. See also Power of AdmiralsQuarters role. The role can be assigned by the `accessManager` role, which is assumed to be governance. As a result, governance was trusted not to assign the role to a malicious address, otherwise the staked funds can be drained. However, in Version 2 the FleetCommanderRewardsManager was refactored and can now no longer withdraw funds to an arbitrary address. This limits the trust requirements of the AdmiralsQuartersRole. It can no longer drain all funds. Governance can also assign the `governor` role. The `governor` should correctly assign rewards to be distributed to stakers and should not add the staking token as a reward token. Aside from this, the governor is untrusted.

The DutchAuctionLibrary is used by the Raft contract. The Raft contract is assumed to use good defaults for auction creation and not allow arbitrary auction parameters. The defaults are set by the Raft's `governor` role. If bad parameters are set, all funds in the Raft contract could be drained. As a result, the `governor` (and roles that can assign the governor role) is fully trusted.

For the dutch auctions, it is assumed that there are at least two competitive bidders monitoring all ongoing auctions.

Tokens used in the system are assumed to be compliant with the ERC20 standard, and not use a very large (e.g. >24) number of `decimals`. The tokens should not implement special behaviors (e.g., rebasing, fees on transfer).

Scope 3

- Users are fully untrusted

- Bearers of the `CURATOR_ROLE` for a particular `FleetCommander` are expected to always act in the best interest of the protocol. In particular, as they are responsible for assessing and managing risks, they are trusted to set the risk management parameters of the fleet with the goal of maximizing the yield and users' fund risk exposure in the different integrated protocols.
- Bearers of the `GUARDIAN_ROLE` in the vesting wallet are expected to always act in the best interest of the recipients of the distributed token. In particular, they are trusted to manage the goals in a fair manner.
- Bearers of the `KEEPER_ROLE` for a particular `FleetCommander` are expected to always act in the best interest of the protocol. In particular, they are trusted to rebalance their fleet's arks and adjust the buffer in order to maximize the return.

4 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

5 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

6 Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

- Security: Related to vulnerabilities that could be exploited by malicious actors
- Design: Architectural shortcomings and design inefficiencies
- Correctness: Mismatches between specification and implementation

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	3

- Missing Token Transfer in SummerVestingWallet Risk Accepted
- Pendle's totalAssets() Jumps Over Expiry Risk Accepted
- Withdrawing From Expired Markets Can Drop the Shares Price Risk Accepted

Low-Severity Findings	4

- Boarding AaveV3Ark Can Fail Acknowledged
- Disembarking Can Only Take Place in Specific Market Conditions Risk Accepted
- Rebalance Cooldown Can Be Circumvented Acknowledged
- Unremovable Reward Tokens Code Partially Corrected

6.1 Missing Token Transfer in `SummerVestingWallet`

Correctness Medium Version 1 Risk Accepted

CS-SMMRFI-EARN-001

When setting a new goal, by calling `SummerVestingWallet.addNewGoal()`, the `goalAmount` of the vesting token gets transferred from the caller (with guardian role) to the vesting wallet.

Setting new goals can also be done during the deployment. However, in the constructor a transfer of the vesting token with an amount equal to sum of all `goalAmounts` is missing in comparison with `addNewGoal()` logic.

Also, the contract has no guarantee that the `timeBaseVestingAmount` token amount will be in the contract after deployment.

Risk accepted:

Summer.fi is aware of this behavior and answers:

```
We rely on the fact that the tokens are transferred to the vesting wallet, after it's constructed
(time based and sum of goal amounts) - in the SummerVestingWalletFactory. Additional change : there
is a new role -`FOUNDATION_ROLE` managed centrally  only this role is allowed to create new vesting
wallets.
```

6.2 Pendle's `totalAssets()` Jumps Over Expiry

Design | Medium | Version 1 | Risk Accepted

CS-SMMRFI-EARN-002

The function `totalAssets()` returns the total asset held by the ark after expiry. However, if called before expiry, a percentage of that is deducted. Therefore, when users deposit into the `FleetCommander` will receive less shares after expiry. The extent to which it influences the shares depends on all assets over all arks, the amount of assets in this Ark, and slippage percentage.

Risk accepted:

Summer.fi is aware of potential issues related to this and states:

```
Assuming a minor fraction of the fleet TVL allocated to Pendle ark,
as well as keepers working in favor of the users ( regarding rollover
or funds withdrawal post expiry) we accept the risk.
```

6.3 Withdrawing From Expired Markets Can Drop the Shares Price

Design | Medium | Version 1 | Risk Accepted

CS-SMMRFI-EARN-003

The governance is allowed to redeem the PT balance from an expired market via `PendlePtOracleArk.withdrawExpiredMarket()`. If the redeemed tokens are not atomically swapped and put back in the fleet without minting new shares, the price of the shares can significantly drop, hurting users.

Risk accepted:

Summer.fi is aware of the potential issue and states:

```
For now no changes are applied, we trust governance would swap the tokens to
fleet asset and send to buffer ark. If that proves to be insufficient, we'll
implement a generic swap method, allowing for passing swap calldata and
transferring swap output tokens to the buffer ark.
```

6.4 Boarding `AaveV3Ark` Can Fail

Design | Low | Version 1 | Acknowledged

CS-SMMRFI-EARN-004

When rebalancing the arks or adjusting the buffer, a strategy can be to board liquidity in the `AaveV3Ark`. But this can fail as Aave can implement a supply cap on its markets, or if the market is frozen or paused.

This can also affect the auction mechanism; as the auction is not able to board the `AaveV3Ark`, no other auction can be started for that `(ark, rewardToken)` pair.

Acknowledged:

Summer.fi is aware of this potential issue and states:

```
We push the responsibility of validating the protocol status on the keeper
```

6.5 Disembarking Can Only Take Place in Specific Market Conditions

Design | Low | Version 1 | Risk Accepted

CS-SMMRFI-EARN-005

The function `PendlePtOracleArk._swapPtForFleetAsset()` succeeds only when the amount of PT can be swapped into at least the amount `_amount` that should be disembarked. It is not guaranteed that this condition is met when the keeper wishes to disembark liquidity and thus could have an effect on the liquidity available at all time.

Risk accepted:

Summer.fi is aware of this potential issue and states:

```
Keeper should only try to disembark at optimal conditions.
Users cant directly withdraw from this ark, if liquidity is
required it's the keepers role to disembark and rebalance to buffer.
```

6.6 Rebalance Cooldown Can Be Circumvented

Design | Low | Version 1 | Acknowledged

CS-SMMRFI-EARN-006

The functions `rebalance()` and `adjustBuffer()` in `FleetCommander` are protected by the `enforceCooldown` modifier to avoid being called too often by the keeper. This cooldown can be reset by redeeming shares or withdrawing assets, even `0`, effectively allowing the keeper to move funds around -when funds allocation didn't or almost didn't change- before they should have been allowed to by the cooldown period.

Acknowledged:

Summer.fi is aware of the issues and states:

```
We leave it as is assuming KEEPER is not malicious and will not try to DOS the fleet. The main
goal of this mechanic is to be able to rebalance a fleet that has been pushed out of balance
by ark withdrawal.
```

6.7 Unremovable Reward Tokens

Correctness | Low | Version 1 | Code Partially Corrected

CS-SMMRFI-EARN-007

The `FleetCommanderRewardsManager.removeRewardToken()` requires that the contract does not hold any amount of the token to remove.

This is very unlikely to happen in practice since dust could be left in the contract due to rounding, unclaimed rewards, or donations to the contract.

As a result, the governor will likely be unable to remove unused reward tokens, thus forcing users to waste gas on them forever.

Code partially corrected:

`removeRewardToken()` now allows removing a token if the balance is less than a dust threshold of 0.0001 token. This allows removing tokens with very small balances. However, it is likely that most tokens will always have a larger balance than this.

It should be expected that most tokens can never be removed as rewards tokens. As a result, only a small number of tokens should be added as rewards.

7 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	5

- Setting Rewards Manager Stores a Different Address Code Corrected
- State Not Updated Before Staking Code Corrected
- Tip Not Collected Code Corrected
- Tipped Shares Not Taken Into Account Code Corrected
- Wrong Order Assumption in Withdrawable Arks Caching Code Corrected

Medium-Severity Findings	10

- Wrong Direction for Buffer Adjustment Checks Code Corrected
- Auctions Can Be Locked With Wrong Payment Token Code Corrected
- Compounding Tip Model Code Corrected
- Default Auction Params May Not Include Fair Token Price Code Corrected
- Disembarking AaveV3Ark Can Fail Code Corrected
- Guardian Can Drain the Vesting Wallet Code Corrected
- Max Rebalance Flows Can Be Circumvented Code Corrected
- Position Token Can Be Swept on Arks Code Corrected
- Unreached Goals Accounted After End of Vesting Code Corrected
- price_oracle Should Be Used to Read EMA Price Code Corrected

Low-Severity Findings	24

- Aave Can Be Withdrawn When Frozen Code Corrected
- Non Team Vesting Cannot Have Goals Code Corrected
- Positive Slippage in _swapPtForFleetAsset Code Corrected
- Calling sweep() on the BufferArk Moves All the Buffered Liquidity Code Corrected
- Code With No Effect Code Corrected
- FleetCommander Shares Can Be Locked in the Vault Code Corrected
- Inconsistent State of rewardTokensList Code Corrected
- Last Array Element Shortcut Code Corrected
- Misleading Function Name Code Corrected
- Misleading Variable Name Code Corrected
- Missing Events Code Corrected
- Missing Getter for Details Code Corrected
- Missing Input Sanitization Code Corrected

- Remove onlyCommander Code Corrected
- Revert Instead of Return Code Corrected
- Setting Duration of Vesting Wallet Code Corrected
- Simplify Redemption Post-Expiry Code Corrected
- Slippage Protection and Price Oracle in PendlePtOracleArk Code Corrected
- Staking Token Can Be a Reward Token Code Corrected
- Storage Not Properly Cleaned After Ark Removal Specification Changed
- Stronger Requirement for Commander Registration Code Corrected
- Tip Stream for address(0) Should Not Be Allowed Code Corrected
- Wrong Specifications Specification Changed
- AdmiralsQuarters Does Not Include a Callpath to getReward() on the StakingRewardsManager Code Corrected

Informational Findings	12

- Code Consistency Code Corrected
- Arbitrary Harvest Data Code Corrected
- FleetCommander Address Could Be Immutable Code Corrected
- Incorrect Dust Threshold Calculation Code Corrected
- Misnaming of Quadratic Decay Function Code Corrected
- Power of AdmiralsQuarters Role Specification Changed
- Readability of Code Code Corrected
- Rewards Balance Check Counts Unclaimed Tokens Code Corrected
- Shares Left Behind When Shaking Fleet Commanders Code Corrected
- Unnecessary Casting Code Corrected
- UnstakeOnBehalfOf Does Not Claim Rewards Code Corrected
- Unused Code Code Corrected

7.1 Setting Rewards Manager Stores a Different Address

Correctness High Version 1 Code Corrected

CS-SMMRFI-EARN-038

In `FleetCommanderConfigProvider.setStakingRewardsManager()`, a `newStakingRewardsManager` address is passed as a parameter, but another reward manager is deployed and set as `config.stakingRewardsManager`. This has the effect of storing an unexpected address as the new staking reward manager.

Code corrected:

The function has been renamed to `updateStakingRewardsManager()` and does not take a parameter. The updated address for the staking manager will be the one returned by `FleetCommanderRewardsManagerFactory.createRewardsManager()`.

7.2 State Not Updated Before Staking

Correctness | High | Version 1 | Code Corrected

CS-SMMRFI-EARN-053

`FleetCommanderRewardsManager.stakeOnBehalfOf()`, and `FleetCommanderRewardsManager.unstakeOnBehalfOf()` never call `_updateReward()`, they use the internal functions of StakingRewardsManagerBase, which do not have the `updateReward` modifier.

As a result, users can increase and decrease their balance without being checkpointed first, leading to incorrect rewards calculations.

In particular, the `userRewardPerTokenPaid` of a new address will be zero when an address stakes for the first time. If the stake function was correct, this value would be updated to the current `rewardPerTokenStored` before increasing the user´s balance. As a result, the user will be eligible for staking rewards as if they had deposited at the creation time of the contract and never claimed their rewards in the meantime.

This allows the following attack:

1. `stakeOnBehalfOf()` using a new address
2. `getReward()` to get rewards for the entire lifetime of the rewards contract
3. `unstake()` to receive the stake back

This attack can be repeated an arbitrary number of times by using a new address each time. As a result, it can be used to drain all rewards tokens held by the contract. In the special case where the staking token is also registered as a rewards token, it could also be used to drain all staked tokens, see Staking token can be a reward token.

Additionally, the `rewardPerTokenStored` calculation will be incorrect, as it relies on `updateReward` being called before any change to the `totalSupply` value.

Code corrected:

The functions in FleetCommanderRewardsManager have been updated to call `_updateReward()` before changing the user balance. This solves the issue.

7.3 Tip Not Collected

Correctness | High | Version 1 | Code Corrected

CS-SMMRFI-EARN-051

The function `FleetCommander.withdrawFromBuffer()` is missing the `collectTip`. This allows users to withdraw their assets without diluting the shares and skip sending the tip to the `TipJar`, effectively enabling them to withdraw more than what the system would expect and escape the fee.

Code corrected:

The `collectTip` modifier was added to `FleetCommander.withdrawFromBuffer()`.

7.4 Tipped Shares Not Taken Into Account

Correctness **High** **Version 1** **Code Corrected**

CS-SMMRFI-EARN-025

When computing assets or shares in the following functions of the `FleetCommander`, the shares corresponding to a potential tip to be collected are not taken into account:

- all the ERC4626 preview functions
- `maxMint()`, `maxBufferWithdraw()`, `maxWithdraw()`, `maxRedeem()`, `maxBufferRedeem()`

This behavior could potentially break systems integrating with Summer Earn Protocol.

Code corrected:

The `totalSupply()` function has been overriden to add a virtual share amount, corresponding to the tip, to the totalSupply when the tip in not collected previously in the callpath.

7.5 Wrong Order Assumption in Withdrawable Arks Caching

Correctness **High** **Version 1** **Code Corrected**

CS-SMMRFI-EARN-032

In the function `FleetCommanderCache._getWithdrawableArksData()`, the `_arksData` coming from `_getArksData()` are sorted by their total assets (ascending). In the following `for` loop, the condition `i == _arksData.length - 1` will make the last ark of the array `withdrawable`, with the assumption that the last ark will be the `BufferArk`. This assumption does not hold because of the arks ordering mentioned above. This can have the effect that a non-withdrawable ark can be cached as withdrawable if it is the last element of the array. In this case, the withdraw/redeem from arks process can be temporarily blocked because the `_disembark()` call without disembarking data will fail for the non-withdrawable ark.

Code corrected:

The sorting in `_getArksData()` has been removed.

7.6 Wrong Direction for Buffer Adjustment Checks

Correctness **Medium** **Version 2** **Code Corrected**

CS-SMMRFI-EARN-036

The function `FleetCommander._validateAdjustBuffer()` must ensure that enough liquidity stays in the buffer after adjustment. In the loop, the direction of the adjustment is always read from `rebalanceData[0]`, for all rebalancing operation touching the `BufferArk`. The code will consider the direction of all buffer-related operations having the same direction as

`rebalanceData[0].toArk == _bufferArkAddress`, which could not even be a buffer-related operation. The effect of this behavior is that `_validateAdjustBuffer()` does not correctly track and check the liquidity left in the buffer after the adjustment.

Code corrected:

The code has been corrected to ensure that the direction of the adjustment is correctly read from the rebalance data at each iteration of the loop, i.e., reads from `rebalanceData[i]`.

7.7 Auctions Can Be Locked With Wrong Payment Token

Design | Medium | Version 1 | Code Corrected

CS-SMMRFI-EARN-027

Auctions for an `(ark, rewardToken)` pair can be initiated with arbitrary payment tokens. If the payment token for an `ark` doesn't correspond to the ark's `config.asset`, the `_board()` will fail as `paymentToken` will be approved but the `ark` expects its `config.asset`.

This has an effect that the auction for the pair `(ark, rewardToken)` can never be settled and it prevents any future auctions for that pair, locking the reward in the ark's position.

Code corrected:

The payment token is queried directly from the target ark.

7.8 Compounding Tip Model

Design | Medium | Version 1 | Code Corrected

CS-SMMRFI-EARN-043

The current model for the tip is a compounding exponential system applied to the number of shares of the vault. If it gets used with systems that have a linear yield growth model, the shares may be losing value faster than the farmed yield, leading to users losing money.

Code corrected:

Summer.fi has changed the tip model from a compounding exponential function to a linear function.

7.9 Default Auction Params May Not Include Fair Token Price

Design | Medium | Version 1 | Code Corrected

CS-SMMRFI-EARN-026

The default auction start and end price are fixed and are applied regardless of the payment and reward token. This can lead to two extreme cases:

1. The starting price is too low for the reward token and the reward token will be sold for less than what it should
2. The final price is still too high for the reward token and will not be bought

---

Code corrected:

Auctions parameters are now set by the curator of the target ark's commander. A new mapping stores the auction parameters for each `(ark, rewardToken)` pairs. The curator is expected to correctly update the parameters based on market conditions.

7.10 Disembarking `AaveV3Ark` Can Fail

Design | Medium | Version 1 | Code Corrected

CS-SMMRFI-EARN-055

If there is not enough liquidity in the target market or the market is paused by Aave, disembarking from `AaveV3Ark` can fail and effectively DOS the redeem/withdraw from arks mechanism, as well as preventing rebalancing from that ark. If this happens, users' funds will be locked until the liquidity crisis or pausing is over, and the vault will be temporarily insolvent.

Code corrected:

The `AaveV3Ark` has been updated with the `_withdrawableTotalAssets()` function. It returns `0` if the market is not active, paused, or frozen, and otherwise returns the minimum between the ark's `aToken` balance and the balance of underlying token in the `aToken` to determine the maximum amount that can be withdrawn from the AaveV3 market.

7.11 Guardian Can Drain the Vesting Wallet

Design | Medium | Version 1 | Code Corrected

CS-SMMRFI-EARN-062

A user holding `GUARDIAN_ROLE` can call `recallUnvestedTokens()` to transfer the unreached goals of the vesting, if the wallet is in team vesting mode. The reason behind it, is that `recallUnvestedTokens()` calls `_calculateUnvestedPerformanceTokens()`, which iterates over the array goal amounts and sums up those for which the goal is not reached. However, during this call path the corresponding `goalAmounts` are not reset to 0. Hence, a second call to `recallUnvestedTokens()` can still accumulate the already claimed unvested amount.

Code corrected:

The code has been updated to reset the `goalAmounts` to 0 during the callpath of `recallUnvestedTokens()`.

7.12 Max Rebalance Flows Can Be Circumvented

Design | Medium | Version 1 | Code Corrected

CS-SMMRFI-EARN-048

The checks for `maxRebalanceOutflow` and `maxRebalanceInflow` in `_validateReallocateAssets()` are done per rebalancing pair and not over the whole rebalancing batch. This allows the keeper to split a rebalancing, overshooting the limits, in smaller parts to circumvent the in/out flow checks.

Code corrected:

The codebase has been updated to keep track of the in- and outflows per ark with a caching mechanism. The rebalancing fails as soon as one of the cumulative flow value for an ark exceeds the maximum allowed.

7.13 Position Token Can Be Swept on Arks

Design | Medium | Version 1 | Code Corrected

CS-SMMRFI-EARN-063

Nothing prevents the position tokens of an ark to be targeted by a `sweep()` (for example, Ark is the owner of `aToken` in case of `AaveV3Ark`). If this happens, the value of the vault's shares can significantly drop, allowing users to buy cheap shares. The price of the shares can also jump if the token is sold in an auction and the payment token is boarded. While the likelihood of this happening is limited since it would need to be triggered by the governor, the impact of such an attack/mistake is significant.

Code corrected:

The sweepable tokens are now whitelisted for each ark. The whitelisting per ark is managed by the curator of the target ark's commander.

7.14 Unreached Goals Accounted After End of Vesting

Correctness | Medium | Version 1 | Code Corrected

CS-SMMRFI-EARN-034

In `SummerVestingWallet._vestingSchedule()`, if end of the vesting is reached, `totalAllocation` is returned (which is all the vesting token held by this contract + the released vesting tokens), although still some goals might be unreached after the end of vesting. The amounts related to those goals are released after the end of the vesting period.

Version 2

The contract does not simply release the `totalAllocation` if the end of the vesting is reached. Instead, it uses the same logic as when the vesting is not ended yet, but this means `timeBasedVested` will be greater than `timeBasedVestingAmount` and thus the missing amount will be taken from the funds allocated for the performance goals.

Code corrected:

The `_calculateTimeBasedVesting` function caps the time based vested amount to the `timeBasedVestingAmount` set during deployment.

7.15 `price_oracle` Should Be Used to Read EMA Price

Design | Medium | Version 1 | Code Corrected

CS-SMMRFI-EARN-033

The function `ema_price()` may return a stale value for the EMA price. To read the EMA price, `price_oracle()` should be used instead.

Code corrected:

The code has been updated to use `price_oracle()` instead of `ema_price()`.

7.16 Aave Can Be Withdrawn When Frozen

Design | Low | Version 2 | Code Corrected

CS-SMMRFI-EARN-047

The `AaveV3Ark` returns `0` as withdrawable assets when the market is frozen, but it is possible to withdraw from a frozen market.

Code corrected:

The code has been corrected to allow withdrawing from a frozen market by returning the withdrawable amount instead of `0`.

7.17 Non Team Vesting Cannot Have Goals

Correctness | Low | Version 2 | Code Corrected

CS-SMMRFI-EARN-050

In the constructor of `SummerVestingWallet` the code reverts if the vesting type is not `InvestorExTeamVesting` and the goals array is not empty. However, there exist only two types of vesting, so the `if-else` callpath will never be taken, as `!InvestorExTeamVesting` is `TeamVesting`, which is taken care of in the first `if` branch. The code should revert if the vesting type is `InvestorExTeamVesting` and some goals are set.

Code corrected:

The constructor has been updated to revert if the vesting type is `InvestorExTeamVesting` and the goals array is not empty.

7.18 Positive Slippage in `_swapPtForFleetAsset`

Design | Low | Version 2 | Code Corrected

CS-SMMRFI-EARN-060

In `_swapPtForFleetAsset`, there is a positive slippage guard, that reverts if `exactPtIn < minPtAmount`.

```
if (exactPtIn < minPtAmount || exactPtIn > maxPtAmount)
    revert InvalidAmount();
```

However, exchanging less PT tokens for the asset is desirable.

Code corrected:

Summer.fi has removed this check.

7.19 Calling `sweep()` on the `BufferArk` Moves All the Buffered Liquidity

Design | Low | Version 1 | Code Corrected

CS-SMMRFI-EARN-035

When `sweep()` is called on the `BufferArk`, the first thing done by the function is boarding again the whole liquidity of the ark. This incurs avoidable gas cost and a misleading `Boarded` event indicating a significant amount of asset entered the system.

Code corrected:

A condition has been added in `Ark.sweep()` to skip boarding the fleet asset if the ark is the `BufferArk`.

7.20 Code With No Effect

Design | Low | Version 1 | Code Corrected

CS-SMMRFI-EARN-030

In `Raft._board()`, a call to `IArk(ark).requiresKeeperData()` is done but the returned value is never read nor used, rendering this call useless and wasting gas.

Code corrected:

The call to `IArk(ark).requiresKeeperData()` was removed.

7.21 FleetCommander Shares Can Be Locked in the Vault

Design | Low | Version 1 | Code Corrected

CS-SMMRFI-EARN-039

The `FleetCommander` overrides the `ERC20.transfer()` function to block all transfers, except if the caller is the rewards manager, or if the destination is the `FleetCommander` itself. If shares are

transferred to the vault (`FleetCommander`), they will be locked as there is no function to transfer them out again.

Code corrected:

The special rule for the `FleetCommander` as a recipient has been removed.

7.22 Inconsistent State of rewardTokensList

Correctness Low Version 1 Code Corrected

CS-SMMRFI-EARN-056

In StakingRewardsManagerBase, reward tokens should be added to the `_rewardTokensList` by calling `notifyRewardAmount()`. This sets its reward duration from zero to a non-zero value.

However, `FleetCommanderRewardsManager.setRewardsDuration()` can also be used on a token that was not added as a rewards token, to set its `rewardsDuration`. If `notifyRewardAmount()` is subsequently used, it will treat it as an already added reward token due the check on `rewardsDuration`, but it will not be in `_rewardTokensList`, meaning that rewards cannot be updated and claimed.

It should be possible to recover from this situation by waiting for the end of the reward period and resetting `rewardsDuration`. Funds in the rewards manager can always be distributed as rewards again.

Code corrected:

A check has been added in `setRewardsDuration()`. Now, only tokens that were properly added can be passed.

7.23 Last Array Element Shortcut

Design Low Version 1 Code Corrected

CS-SMMRFI-EARN-037

In the `FleetCommanderCache`, the functions `_withdrawableTotalAssets()` and `_getWithdrawableArksData()` have a special condition in their loop for the last element of the array. This check for the buffer ark is however redundant (as the buffer ark has withdrawable asset equal to its total assets and will be considered as a withdrawable ark if there are any assets kept in it).

Code corrected:

In both of the above mentioned functions, the special check for the last element of the array has been removed. Being withdrawable is checked only via the withdrawable assets of an ark being non-zero.

7.24 Misleading Function Name

Design Low Version 1 Code Corrected

CS-SMMRFI-EARN-052

For the sake of code readability and maintainability, variable names should mirror the meaning of the value they hold.

1. the function `PendlePtOracleArk._redeemFleetAssetFromPtPostExpiry()` is redeeming market asset, not fleet asset
2. the function `CurveExchangeRateProvider._applyEmaRange()` is used on EMA price, but also on spot price

---

Code corrected:

1. FIX
2. FIX

7.25 Misleading Variable Name

Design | Low | Version 1 | Code Corrected

CS-SMMRFI-EARN-058

For the sake of code readability and maintainability, variable names should mirror the meaning of the value they hold.

- In `AuctionManagerBase`, the variable `nextAuctionId` should hold the id of the next auction to be started, but the actual id of the next auction is `nextAuctionId+1`.
- In `FleetCommanderConfigProvider`, the constant `MAX_REBALANCE_OPERATIONS` is used as initial value for `maxRebalanceOperations`, but then `maxRebalanceOperations` can be set to a value greater than `MAX_REBALANCE_OPERATIONS`.

Code corrected:

- FIX: The `nextAuctionId` variable has been renamed to `currentAuctionId`
- FIX: The `MAX_REBALANCE_OPERATIONS` is now used to sanitize the new value of `maxRebalanceOperations`

7.26 Missing Events

Design | Low | Version 1 | Code Corrected

CS-SMMRFI-EARN-042

Events should be emitted when an important state change happens on-chain. Ideally, an observer should be able to reconstruct the state by observing the events. The following state changes should emit an event:

Scope 1:

1. the constructor of `AuctionManagerBase` should emit the `AuctionDefaultParametersUpdated()` event
2. the constructor of `FleetCommanderConfigProvider` should emit the `ArkAdded()` event for the `BufferArk`
3. the constructor of `FleetCommanderPausable` should emit the `MinimumPauseTimeUpdated()` event

Scope 3:

1. When adding new goal, through either `addNewGoal` or the constructor of `SummerVestingWallet`
2. When marking a goal as reached (via `markGoalReached()`)
3. When transferring vesting tokens for unreached goals in `recallUnvestedTokens()`
4. In the constructor of `ExchangeRateProvider()` after setting the `basePrice` the corresponding event `BasePriceUpdated` does not get emitted

---

Code corrected:

Scope 1:

1. FIX: the default parameters have been removed
2. FIX: the constructor has been updated to emit the `ArkAdded()` event for the `BufferArk`
3. FIX: the constructor has been updated to emit the `MinimumPauseTimeUpdated()` event

Scope 3:

1. FIX
2. FIX
3. FIX
4. FIX

7.27 Missing Getter for Details

Design | Low | Version 1 | Code Corrected

CS-SMMRFI-EARN-029

The contract `ArkConfigProvider` implements getters for all the fields of the `ArkConfig` struct except for `details`.

Code corrected:

A getter for `details` has been added.

7.28 Missing Input Sanitization

Design | Low | Version 1 | Code Corrected

CS-SMMRFI-EARN-059

Scope 1:

1. In the constructor of `ArkConfigProvider`, `_params.maxDepositPercentageOfTVL` is not checked to be `<= 100%`
2. In `ArkConfigProvider.setMaxDepositPercentageOfTVL()`, the new value is never checked to be `<= 100%`
3. In `CooldownEnforcer._updateCooldown()`, the new value is never checked to be in a reasonable range

4. In the constructor of `Tipper`, `initialTipRate` is never checked to be in a reasonable range
5. In `AuctionManagerBase`, the default auction parameters are never sanitized in the `constructor` and when updated with `_updateAuctionDefaultParameters()`
6. In `TipJar` the value of `lockedUntilEpoch` is never sanitized in `addTipStream()` or `updateTipStream()`.
7. In `FleetCommanderPausable`, the `_newMinimumPauseTime` is not checked to be in a reasonable range when updating the minimum pause time.
8. In `FleetCommanderConfigProvider._addArk()`, the new ark's asset is not enforced to match the fleet's underlying asset.
9. The `FleetCommander` allows users to mint, deposit, redeem and withdraw `0` values. This consumes gas and emits unnecessary events. Disallowing `0` values can also protect the users against redeeming their shares for `0` assets if the vault is manipulated for example.
10. In `FleetCommanderConfigProvider.setMaxRebalanceOperations()`, the input variable `newMaxRebalanceOperations` is not enforced to be at most `MAX_REBALANCE_OPERATIONS`.

Scope 3:

1. In the constructor of `CurveExchangeRateProvider` there is no sanity check enforcing `_curveSwap` is not `address(0)`.
2. The function `ExchangeRateProvider._setBasePrice()` does not enforce the new `basePrice` to be non-zero
3. The constructor of `CurveExchangeRateProvider` also does not enforce that `baseToken` belongs to the `curveSwap`.
4. In the constructor of `SummerVestingWallet`, `_token` is not enforced to be non-zero.
5. In the constructor of `PendlePtOracleArk`, the `router`, `market`, and `oracle` are missing a zero address check
6. In the constructor of `SummerVestingWallet`, it is possible to set goals even if the vesting type is not `TeamVesting`
7. The function `SummerVestingWallet.addNewGoal()` allows the guardian to set goals even if the vesting type is not `TeamVesting`
8. In the function `PendlePtOracleArk._swapFleetAssetForPt()`, `input.tokenIn` should be enforced to be equal to `config.asset`
9. In the function `PendlePtOracleArk._swapFleetAssetForPt()`, `input.netTokenIn` should be enforced to be equal to `_amount`. If not equal, the price of shares may drop because some fleet asset will stay in the ark.
10. In the function `PendlePtOracleArk._swapPtForFleetAsset()`, `output.tokenOut` should be enforced to be equal to `config.asset`
11. In `PendlePtOracleArk.setOracleDuration`, the new `oracleDuration` should be below a certain threshold that makes sense for the system

---

Code corrected:

Scope 1:

1. FIX: the `_params.maxDepositPercentageOfTVL` is checked to be in the `[0%, 100%]` range
2. FIX: the new value for `maxDepositPercentageOfTVL` is checked to be in the `[0%, 100%]` range

3. FIX: the cooldown value is checked to be between 1 minute and 1 day in the constructor and in the update function
4. FIX: the initial tip rate is enforced to be at most 5% in the constructor and in the update function
5. FIX: the default auction parameters have been removed
6. FIX: the function has been updated to enforce a maximum locking time of 750 days in both functions
7. FIX: the minimum pause time is enforced to be at least 2 days in the constructor and the update function
8. FIX: the `_addArk()` function has been updated to check that the ark's asset and the fleet's asset match
9. FIX: zero checks have been added to the mint, deposit, redeem and withdraw functions
10. FIX: the `newMaxRebalanceOperations` checked to be at most `MAX_REBALANCE_OPERATIONS`

Scope 3:

1. FIX
2. FIX
3. FIX
4. FIX
5. FIX
6. FIX
7. FIX
8. FIX
9. FIX
10. FIX
11. FIX

7.29 Remove `onlyCommander`

Design | Low | Version 1 | Code Corrected

CS-SMMRFI-EARN-044

The function `ArkConfigProvider.unregisterFleetCommander()` has the `onlyCommander`, given that only the current commander can unregister himself, this modifier is redundant.

Code corrected:

Summer.fi has removed this redundant modifier.

7.30 Revert Instead of Return

Design | Low | Version 1 | Code Corrected

CS-SMMRFI-EARN-028

In the function `TipJar._shake`, the function reverts whenever `shares==0` or `withdrawnAssets==0`.

Code corrected:

In `TipJar._shake()` Summer.fi has changed the code to return early by emitting `TipJarShaken` events with `totalDistributed` parameter set to 0.

7.31 Setting Duration of Vesting Wallet

Design | Low | Version 1 | Code Corrected

CS-SMMRFI-EARN-046

In `SummerVestingWallet`, `durationSeconds` is a constructor's input parameter. In `_calculateTimeBasedVesting()`, however, it is assumed that the duration of the vesting is 2 years (8 quarters). Therefore, it is better to hardcode the duration of vesting to 2 years to make the code consistent and avoid any mistake upon deployment.

Code corrected:

The duration of the vesting is hardcoded to 2 years in the constructor of `SummerVestingWallet`.

7.32 Simplify Redemption Post-Expiry

Design | Low | Version 1 | Code Corrected

CS-SMMRFI-EARN-041

In `_redeemFleetAssetFromPtPostExpiry()`, the two-step redemption of firstly redeeming PY to SY and then redeeming SY can be simplified by using `IPAllActionV3.redeemPyToToken()`.

Code corrected:

Summer.fi has replaced the two-step redemption with a simple call:

```
IPAllActionV3(router).redeemPyToToken(
    address(this),
    address(YT),
    ptAmount,
    tokenOutput)
```

7.33 Slippage Protection and Price Oracle in `PendlePtOracleArk`

Design | Low | Version 1 | Code Corrected

CS-SMMRFI-EARN-057

The `shouldTrade` and `shouldBuy` modifiers call `_shouldTrade()` to ensure the EMA price fetched from the Curve stableswap pool is in a preset range before a swap is executed on an arbitrary system. In the case where either the curve pool or the arbitrary system (AMM) is manipulated, the `_shouldTrade`

check may not be sufficient to protect the system against slippage, as the Curve EMA price might not reflect the swap price.

Code corrected:

Summer.fi has added an extra level of protection by considering the expected amount of PT by relying on the conversion rate between PT and the underlying asset provided by the oracle. A small `slippagePercentage` is however tolerated.

7.34 Staking Token Can Be a Reward Token

Security Low Version 1 Code Corrected

CS-SMMRFI-EARN-061

In `StakingRewardsManagerBase`, the governor can add the staking token itself as a reward token. This could lead to insolvency by redistributing the staked tokens held in the contract.

The following check is ineffective in this scenario since the contract holds the users' tokens in its balance:

```
uint256 balance = rewardToken.balanceOf(address(this));
if (rewardTokenData.rewardRate > balance / rewardTokenData.rewardsDuration)
    revert ProvidedRewardTooHigh();
```

Code corrected:

The `notifyRewardAmount` function in FleetCommanderRewardsManager now overrides the StakingRewardsManagerBase and checks that the staking token cannot be added as a reward token.

7.35 Storage Not Properly Cleaned After Ark Removal

Design Low Version 1 Specification Changed

CS-SMMRFI-EARN-054

When an ark is removed from the `FleetCommander`, it is properly removed from the active arks but not from the `isArkWithdrawable` mapping if necessary.

Specification changed:

The `FleetCommander` does not store whether an ark is withdrawable or not anymore. Instead, the ark will provide this information when necessary.

7.36 Stronger Requirement for Commander Registration

Design Low Version 1 Code Corrected

CS-SMMRFI-EARN-045

In the current implementation, the `FleetCommanderConfigProvider` is responsible for checking that a potential new ark does not already have a registered commander. This check can be done directly in the ark, i.e. have `ArkConfigProvider.registerFleetCommander()` requiring that `config.commander` is `address(0)`.

The check can be extended to `ArkConfigProvider.unregisterFleetCommander()` checking that the current `config.commander` can unregister (is the caller).

It would be a stronger condition, as in this way, if there is already an active commander for an ark, it will not get overwritten by another address holding `COMMANDER_ROLE`.

Code corrected:

Summer.fi has corrected this issue by adding two checks at the beginning of the two aforementioned functions:

```
function registerFleetCommander() external onlyCommander {
    if (config.commander != address(0)) {
        revert FleetCommanderAlreadyRegistered();
    }
    config.commander = msg.sender;
    emit FleetCommanderRegistered(msg.sender);
}

function unregisterFleetCommander() external onlyCommander {
    if (_msgSender() != config.commander) {
        revert FleetCommanderNotRegistered();
    }
    config.commander = address(0);
    emit FleetCommanderUnregistered(msg.sender);
}
```

7.37 Tip Stream for `address(0)` Should Not Be Allowed

Design | Low | Version 1 | Code Corrected

CS-SMMRFI-EARN-040

If `address(0)` is added as a tip stream recipient, it becomes impossible to call `removeTipStream()` or `updateTipStream()` on it because of `_validateTipStream()`. Through the function `addTipStream()` on can still update the allocation for `address(0)`, but it is not possible to reset its `allocation` to zero (because of the check in `_validateTipStreamAllocation()`).

Code corrected:

Input sanitization has been added to forbid `address(0)` as tip stream recipient.

7.38 Wrong Specifications

Correctness | Low | Version 1 | Specification Changed

Non-exhaustive list of wrong or incomplete specifications:

1. The specification for the `ProtocolAccessManaged` contract corresponds to the `ProtocolAccessManager`.
2. The constructor of `ProtocolAccessManager` specifies that the `governor` address should receive all three `GOVERNOR_ROLE`, `GUARDIAN_ROLE`, `DEFAULT_ADMIN_ROLE`, but only `GOVERNOR_ROLE` is given.
3. The specification for `IArkAccessManaged` mentions `ArkAccessControl` instead of `IArkAccessManaged`
4. The specification of `Raft` mentions it should manage auctions for the harvested rewards as well as the buy-and-burn mechanism, but the `Raft` only manages the auctions for the harvested rewards.
5. The natspec of `FleetCommanderConfigProvider` is missing a title.
6. `FleetCommander.rebalance()` mentions "Validate that no operations are moving to or from the bufferArk" above the call to `_validateReallocateAllAssets()`. This, however, corresponds to the call to `_validateRebalance()`.
7. The constant `DEFAULT_MAX_REBALANCE_OPERATIONS` is never used, in spite of being mentioned in `IFleetCommander`:

```
@dev The number of operations in a single adjustBuffer call is limited to DEFAULT_MAX_REBALANCE_OPERATIONS
```

Version 2

1. The specs of `ProtocolAccessManaged` are missing the description of the `ADMIRALS_QUARTERS_ROLE`
2. The specs of `ProtocolAccessManager` are missing the description of the `ADMIRALS_QUARTERS_ROLE`
3. The specs of `FleetCommanderCache._withdrawableTotalAssets()` mention the "correctness of the isWithdrawable function" instead of the `withdrawableTotalAssets` function
4. The specs of `FleetCommanderCache._getWithdrawableArksData()` mention the "the isWithdrawable function" instead of the `withdrawableTotalAssets` function
5. The specs of `FleetCommanderConfigProvider.onlyActiveArk()` specify "If the arkAddress is the buffer ark, it will not revert". But the `onlyActiveArk()` function will revert for the buffer ark.
6. The specs of `FleetCommanderCache._getArksData()` mention the sorting mechanism, but sorting has been removed from the function

---

Specification changed:

1. FIX: the specs have been updated to match `ProtocolAccessManaged`
2. FIX: the specs have been updated to match the implementation
3. FIX: the specs title has been changed to `IArkAccessManaged`
4. FIX: the specs have been updated to match the implementation
5. FIX: the specs title has been added
6. FIX: the comment has been removed and the implementation of `_validateReallocateAllAssets()` updated
7. FIX: `DEFAULT_MAX_REBALANCE_OPERATIONS` has been removed

Version 2

1. FIX: the specs have been updated to include `ADMIRALS_QUARTERS_ROLE`
2. FIX: the specs have been updated to include `ADMIRALS_QUARTERS_ROLE`
3. FIX: the specs have been updated to mention `withdrawableTotalAssets` instead of `isWithdrawable`
4. FIX: the specs have been updated to mention `withdrawableTotalAssets` instead of `isWithdrawable`
5. FIX: the specs have been updated to match the implementation
6. FIX: the specs have been updated to match the implementation

7.39 `AdmiralsQuarters` Does Not Include a Callpath to `getReward()` on the `StakingRewardsManager`

Design Low Version 1 Code Corrected

CS-SMMRFI-EARN-031

Although the `AdmiralsQuarters` can wrap users' calls to the staking and unstaking functionalities of the staking rewards manager, it does not include the `getReward()` function.

Code corrected:

`AdmiralsQuarters.unstakeAndWithdrawAssets()` introduces a new parameter, namely `claimRewards`, which when set, makes the rewards manager get rewards for the caller when unstaking.

7.40 Code Consistency

Informational Version 2 Code Corrected

CS-SMMRFI-EARN-016

Across the codebase `Constants.MAX_UINT256` is usually used to represent `type(uint256).max`, except in `TipJar._shake()`, `AaveV3Ark._harvest()`, and the constructor of `FleetCommanderConfigProvider`, where `type(uint256).max` is used directly.

Code corrected:

In all of the aforementioned cases `Constants.MAX_UINT256` is used to represent `type(uint256).max`.

7.41 Arbitrary Harvest Data

Informational Version 1 Code Corrected

CS-SMMRFI-EARN-024

The `harvest()` function of the `Raft` can be called with arbitrary `rewardData`. It is important to sanitize that data in the arks using it, otherwise an attacker may be able abuse this data to steal reward tokens.

Code corrected:

The `harvest()` function has been updated with a `onlyGovernor` modifier and is thus now permissioned.

7.42 FleetCommander Address Could Be Immutable

Informational | Version 1 | Code Corrected

CS-SMMRFI-EARN-013

The FleetCommanderRewardsManager defines the `fleetCommander` address as a state variable.

It is only set once in the constructor and never updated, so it could be marked as immutable to save gas.

Code corrected:

The `fleetCommander` address has been marked as immutable.

7.43 Incorrect Dust Threshold Calculation

Informational | Version 1 | Code Corrected

CS-SMMRFI-EARN-014

In `StakingRewardsManagerBase.removeRewardToken()`, the comment states that the dust threshold should be 0.01% of one token. However, in the case where the token's decimals are known and are more than 4, It is computed as `100 * (10 ** (decimals - 4))`, which is 1% of one token.

Code corrected:

The dust threshold is now correctly computed as `10 ** (decimals - 4)`.

7.44 Misnaming of Quadratic Decay Function

Informational | Version 1 | Code Corrected

CS-SMMRFI-EARN-018

The DecayFunction library provides a DecayType called `Exponential` decay. However, the provided functionality is actually a quadratic decay function, which has different properties.

To avoid confusion, the DecayType should be renamed.

Code corrected: `Exponential` has been renamed to `Quadratic`.

7.45 Power of AdmiralsQuarters Role

Informational | Version 1 | Specification Changed

CS-SMMRFI-EARN-020

The AdmiralsQuartersRole in FleetCommanderRewardsManager is a trusted role that is expected to be given to the AdmiralsQuarters contract by governance. However, governance can in principle give this role to any address.

The address with the AdmiralsQuartersRole can call `unstakeOnBehalfOf()` to unstake any user's tokens from the FleetCommanderRewardsManager and give them to any address. As a result, it can drain all staked funds.

This role is powerful and should only be given to trusted contracts.

Specification changed:

In Version 2, the admiral's quarters can unstake users, but cannot recover their principal. As a result, this role is much less powerful than in Version 1.

7.46 Readability of Code

Informational | Version 1 | Code Corrected

CS-SMMRFI-EARN-019

`AdmiralsQuarters.swap()` accomplishes the following two checks:

```
if (
    address(fromToken) == address(0) || address(toToken) == address(0)
) {
    revert InvalidToken();
}
if (assets == 0) revert ZeroAmount();
```

which can be done with the internal functions `_validateToken()` and `_validateAmount()` respectively.

Code corrected:

Summer.fi has used `_validateToken()` and `_validateAmount()` in the aforementioned case.

7.47 Rewards Balance Check Counts Unclaimed Tokens

Informational | Version 1 | Code Corrected

CS-SMMRFI-EARN-023

In StakingRewardsManagerBase, the `notifyRewardAmount()` function does a balance check to ensure that the reward amount is present in the contract.

```
uint256 balance = rewardToken.balanceOf(address(this));
if (rewardTokenData.rewardRate > balance / rewardTokenData.rewardsDuration)
    revert ProvidedRewardTooHigh();
```

This check uses the entire balance of the contract. This means that unclaimed tokens that are allocated to users are counted as part of the balance. This means the check is loose, and may overestimate the amount of rewards that are available to be distributed. This could lead to some users not being able to claim all their rewards.

This is only a problem if the governor chooses a reward amount that is too high.

Code corrected:

In Version 2, the code pulls tokens from the governor directly instead of checking its balance.

7.48 Shares Left Behind When Shaking Fleet Commanders

Informational | Version 1 | Code Corrected

CS-SMMRFI-EARN-021

In `TipJar._shake()`, the number of shares to redeem are cached before `redeem()` and thus `collectTip()` are called, meaning that the redeemed shares will be less than the actual balance of `TipJar` at the time of redemption, if `collectTip()` was not called in the same block. The whole balance could be redeemed by passing `MAX_UINT256`. The shares are not lost as they can be redeemed on the next shake.

Code corrected:

The `_shake()` function was updated to redeem the maximum available shares after tip collection.

7.49 Unnecessary Casting

Informational | Version 1 | Code Corrected

CS-SMMRFI-EARN-017

- In the function `FleetCommander._validateReallocateAssets()`, there are some redundant castings of address to address:

```
if (address(fromArk) == address(0)) {
    revert FleetCommanderArkNotFound(fromArk);
}

if (!isArkActive(address(toArk))) {
    revert FleetCommanderArkNotActive(toArk);
}

if (!isArkActive(address(fromArk))) {
    revert FleetCommanderArkNotActive(fromArk);
}
```

- In the functions `PendlePtOracleArk._redeemFleetAssetFromPtPostExpiry()` and `PendlePtOracleArk._depositMarketAssetForPt()`, there are some redundant castings of address to address:

Example:

```
uint256 tokensToRedeem = IStandardizedYield(SY).previewRedeem(
    address(marketAsset),
    syBalance
);
IStandardizedYield(SY).redeem(
    address(this),
    syBalance,
    address(marketAsset),
    tokensToRedeem,
    false
);
```

Code corrected:

The unnecessary castings have been removed. Furthermore, naming of the function `_redeemFleetAssetFromPtPostExpiry` has been changed to `_redeemMarketAssetFromPtPostExpiry()`.

7.50 UnstakeOnBehalfOf Does Not Claim Rewards

Informational **Version 1** **Code Corrected**

CS-SMMRFI-EARN-015

When a user's stake is unstaked on their behalf by the AdmiralsQuarters contract, their rewards are not claimed.

This means the user will need to separately call the `getReward()` themselves to claim their staking rewards.

Code corrected:

In Version 2, the AdmiralsQuarters can pass the `_claimRewards` flag to `unstakeAndWithdrawOnBehalfOf` to claim the rewards on behalf of the user in the same call.

7.51 Unused Code

Informational **Version 1** **Code Corrected**

CS-SMMRFI-EARN-022

The following parts of codebase are never used:

1. error `InvalidSwapType`
2. error `LowerEmaNotLessThanUpperEma`
3. error `UpperEmaNotGreaterThanLowerEma`
4. error `InvalidMarketExpiry`

Code corrected:

Summer.fi has removed these unused error definitions.

8 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

8.1 Gas Optimizations

Informational **Version 1** **Code Partially Corrected**

CS-SMMRFI-EARN-008

Scope 1:

1. The cached data in `FleetCommander` is computed twice if `redeem()` or `withdraw()` are called. Cached data is computed during calls to `redeem`, `withdraw`, `redeemFromBuffer`, `redeemFromArks`, `withdrawFromBuffer`, `withdrawFromArks` and `redeem/withdraw` are calling `redeem/withdrawFromBuffer/Arks`.
2. In the `FleetCommander`, when withdrawing or redeeming only from the `BufferArk`, the withdrawal data doesn't need to be cached.
3. The function `FleetCommander.withdrawFromBuffer()` is missing a data caching modifier.
4. In the `FleetCommander`, the functions `totalAssets()` and `withdrawableTotalAssets()` could check whether the data is cached before using `getArks()` which will read from storage multiple times.
5. In the `FleetCommander`, the functions `pause/unpause()` have the modifier `whenNotPaused/whenPaused`, but this this redundant since the internal `_pause/_unpause()` functions already have the modifiers.
6. In `ProtocolAccessManaged`, the storage variable `_accessManager` can be immutable.
7. In `ArkConfigProvider`, the `address(0)` check for `_params.configurationManager` done in the constructor is redundant with the check done in the constructor of `ConfigurationManaged`.
8. In multiple functions of `Ark` (`sweep()`, `board()`, `disembark()`, `move()`), `config.asset` can be cached to avoid reading the storage too often.
9. In `Ark.sweep()`, the call `IFleetCommander(config.commander).getConfig().bufferArk` reads the whole struct from storage and loads it in memory. This call can be replaced by `IFleetCommander(config.commander).bufferArk()`.
10. The static roles of `ProtocolAccessManager` can be set as constants in `ProtocolAccessManaged` as well to avoid having to call the `_accessManager` to read the role.
11. The external calls to `this` are unnecessary and can be replaced by internal calls or storage access:
    1. `this.commander()` in `Ark.board()`
    2. `this.getConfig()` in `FleetCommander.transfer()`
    3. `this.totalAssets()` in `FleetCommander.getEffectiveArkDepositCap()`
12. In `FleetCommander._validateBufferExcessFunds`, `config.minimumBufferBalance` can be cached to avoid repetitive storage access.
13. In `FleetCommander._validateBufferArkNotInvolved`, `config.bufferArk` can be cached in memory to avoid repetitive storage access.

14. In `FleetCommanderCache._getWithdrawableArksData`, the call to `_sortArkDataByTotalAssets` is unnecessary as sorting is already performed by `_getArksData`.
15. In `FleetCommanderConfigProvider._addArk()`, the call `IArk(ark).getConfig().commander` can be replaced by `IArk(ark).commander()` to avoid reading the whole `Config` struct from storage and load it in memory.
16. In `AaveV3Ark._harvest()`, `claimRewards()` can be used in place of `claimRewardsToSelf()` to send the reward token directly to the raft, making it one transfer instead of two.
17. In `TipJar.addTipStream()`, when calling `_validateTipStreamAllocation`, the `currentAllocation` can be set to `0` as the stream should not exist, saving a storage read.
18. In `TipJar`, the total allocation can be tracked with a storage variable instead of reading multiple storage slots when `getTotalAllocation()` is called.
19. In `Tipper._accrueTip`, the elapsed time check can be moved higher in the function as it is likely to be triggered more often than a `0%` tip rate.
20. In `Raft._board()`, the whole `DutchAuctionLibrary.Auction` is read from storage but only the `paymentToken` field is needed. A direct field access would save a significant amount of gas.
21. In `Raft` contract the two functions `getAuctionInfo()` and `getObtainedTokens()` return state variables that are public and hence have public getters generated by the compiler.
22. The function `CooldownEnforcer._setLastActionTimestamp()` is always called with `0`. A dedicated "reset" function simply setting `0` would save gas.
23. The `deposit()` function with referral code has the `whenNotPaused` modifier, but this is already enforced in the `deposit()` function without referral code

Version 2

1. In `Raft._startAuction()`, the call `IArk(ark).getConfig().asset` can be replaced by `IArk(ark).asset()` to avoid reading the whole `Config` struct from storage and load it in memory.
2. The function `Tipper.previewTip()` declares and initializes the `tippedShares` variable twice
3. The function `FleetCommander._reallocateAssets()` has a return value of `uint256`, which is never used.

Scope 3:

1. In the function `SummerVestingWallet._vestingSchedule()`, the cliff duration can be added in the first condition to return early when the contract is still in the cliff period
2. In `SummerVestingWallet._calculateTimeBasedVesting()`, the calculation of `elapsedQuarters` can be simplified to:

```
uint256 elapsedQuarters = (timestamp - start()) / QUARTER;
```

3. `SummerVestingWallet._calculateUnvestedPerformanceTokens()`, iterates once over the array to calculate `totalPerformanceTokens`. Then, it calls `_calculatePerformanceBasedVesting()` which iterates once again over the array to sum those goal amounts up, for which the goal has been reached. Then subtracts the two from each other. This can be done in one iteration by summing up only those goal amounts for which the goal has not been reached yet.
4. Furthermore, `totalPerformanceTokens` can be cached and updated only when `recallUnvestedTokens()`, or `addNewGoal()` get called.

5. `AdmiralsQuarters.stake()` fetches the rewards manager from fleet and then validates this address through `_validateRewardsManager()`. However, `stakingRewardsManager` can never be 0 in the fleet. Similar argument holds in `unstakeAndWithdrawAssets()`.
6. The function `AdmiralsQuarters.enterFleet()` can simply use the `fleet.asset()` instead of having `inputToken` as parameter
7. The local variable `underlyingAmount` is not used in `AdmiralsQuarters.moveFromERC4626ToAdmiralsQuarters()`.
8. In `PendlePtOracleArk/CurveExchangeRateProvider` the following variables can be defined as immutable:
    1. `marketAsset`
    2. `router`
    3. `configTokenDecimals`, this variable is also unnecessarily redefined every time rollover occurs
    4. `marketAssetDecimals`, this variable is also unnecessarily redefined every time rollover occurs
    5. `curveSwap`
    6. `baseToken`
9. The constructor of `PendlePtOracleArk` initializes some variables twice:
    1. `curveSwap` through assignment in the constructor and in the `CurveExchangeRateProvider` constructor
    2. `market` through assignment in the constructor and `_updateMarketAndTokens()`
    3. `marketExpiry` through `_updateMarketData()` and `_updateMarketAndTokens()`, which also calls `_updateMarketData()`
10. In the function `PendlePtOracleArk._swapFleetAssetForPt()`, the case `if (this.isMarketExpired())` is caught by the `shouldBuy` modifier
11. Generally, to access the state variables inside a contract, calling them through `this` increases the gas consumption, as it instead of solely reading the state variable, makes an external call to the contract itself. Examples are:
    1. `this.isMarketExpired()` in `PendlePtOracleArk`
    2. `this.nextMarket()` in `PendlePtOracleArk`
    3. `this.validateAndDecodeSwapForPtParams()` in `PendlePtOracleArk`
    4. `this.validateAndDecodeSwapPtForTokenParams()` in `PendlePtOracleArk`
12. `_validateToken()` and `_validateRewardsManager()` implement the same functionality.
13. In `ProtectedMulticall`, if the context for `_msgSender` is always `msg.sender`, the `context` can be removed

Version 2

1. The storage variable `_totalPerformanceTokens` in `SummerVestingWallet` is updated but never read, it might not be needed at all.
2. In the constructor of `SummerVestingWallet`, the check on `_vestingType` in the else-if statement is redundant, as there exist only two variations of the vesting type: `TeamVesting` and `InvestorExTeamVesting`.

Code partially corrected:

Scope 1:

1. FIX
2. FIX
3. FIX
4. FIX
5. FIX: the `whenNotPaused/whenPaused` modifiers have been removed from the `pause()/unpause()` functions
6. FIX: the `_accessManager` storage variable is now immutable
7. FIX: the redundant check in the constructor of `ArkConfigProvider` has been removed
8. FIX: the value of `config.asset` is now cached.
9. FIX: the `sweep()` function has been updated to use a direct call to `IFleetCommander.bufferArk()`
10. FIX: the roles have been copied in `ProtocolAccessManaged`
11. NOFIX
12. FIX: the value of `config.minimumBufferBalance` is now cached
13. FIX: the function `FleetCommander._validateBufferArkNotInvolved` has been removed
14. FIX: sorting in `_getArksData()` was removed, so sorting in `_getWithdrawableArksData()` is now needed
15. FIX: the logic for commander check was moved into the `Ark`
16. FIX
17. FIX
18. NOFIX
19. FIX: the logic of `_accrueTip()` has been updated and does not support the proposed gas saving anymore
20. FIX
21. FIX: both `getAuctionInfo()` and `getObtainedTokens()` are removed
22. FIX: the function has been changed to `_resetLastActionTimestamp`
23. FIX

Version 2

1. FIX: `IArk(ark).getConfig().asset` has been replaced with `IArk(ark).asset()`
2. FIX
3. FIX

Scope 3:

1. FIX: the cliff has been added in the first condition
2. FIX: the calculation of `elapsedQuarters` has been simplified
3. FIX: the calculation of `totalPerformanceTokens` has been optimized
4. FIX: the variable has been removed

5. FIX: the rewards manager is not validated anymore
6. FIX: the function now uses the fleet's asset
7. FIX: the unused variable has been removed
8. FIX: the variables have been defined as immutable
9. FIX: the redundant initializations have been removed
10. FIX: the redundant check has been removed
11. NOFIX
12. FIX: `_validateRewardsManager()` has been removed
13. NOFIX

Version 2

1. FIX
2. FIX

8.2 Read-only Reentrancy on TotalSupply

Informational **Version 1** **Risk Accepted**

CS-SMMRFI-EARN-009

In `StakingRewardsManagerBase`, `stake()` and `unstake()` do not defend against read-only reentrancy. If a token with transfer hooks (such as ERC-777) is used as the staking token, the pre-transfer hook can be used to make the contract report an arbitrarily low total supply or an increased total supply for the duration of the transaction.

If a third party protocol queries the total supply onchain, it might be vulnerable to read-only reentrancy attacks.

The FleetCommander token that is intended to be used as the staking token does not exhibit this behavior. It would only be an issue if a different staking token is used.

Risk accepted:

Summer.fi responded:

```
We are going to use Fleet tokens and wrapped (OZ ERC20 Wrapper) Summer token only.
```

8.3 Removal of Arks Can Be DOSed

Informational **Version 1** **Acknowledged**

CS-SMMRFI-EARN-010

One of the checks conducted when an ark is removed from a fleet commander is `ark.totalAssets == 0`. This check can unexpectedly fail in the following cases:

- the position managed by the ark is represented by a rebasing token, e.g. `aToken`. In this case, some dust can remain in the ark after disembarking the whole amount from the integrated system.
- an attacker can send some tokens to the ark after it is disembarked and before it is removed

In both cases, the ark cannot be removed from the fleet commander.

Acknowledged:

Summer.fi has replied with:

> To remove the ark it's recommended to empty it using forceRebalance and exit in an atomic transaction (by governance). As for rebasing token case
>
> ```
> if(data.amount == Constants.MAX_UINT256) {
>     amout = fromArk.totalAssets();
> } else {
>     amount = data.amount;
> }
> ```
>
> We assumed that this check on rebalance will protect the ark from being left with dust, since it will always get the amount of token in the current block. But indeed that might be a problem with an ark using swaps in and out of the ark asset. In this case we might add a configurable (by curator) dust limit, allowing removal of an ark with assets within dust limit.

It is important that future arks have a way to be fully emptied before removal.

8.4 Unsafe Contract Dependency Graph Assumption

Informational **Version 1** **Acknowledged**

CS-SMMRFI-EARN-011

The modifiers `onlyAuthorizedToBoard()` and `onlyRaft()` in `ArkAccessManaged` rely on the fact that a more derived contract will implement the `IConfigurationManaged` interface without guarantee. It does not pose any issue for the current codebase, but can be problematic if the `ArkAccessManaged` contract is used in a different setup.

Acknowledged:

Summer.fi is aware of this issue.

8.5 buyTokens Can Revert From Frontrunning

Informational **Version 1** **Acknowledged**

CS-SMMRFI-EARN-012

The `buyTokens` function in DutchAuctionLibrary must buy the exact number of tokens specified. If the amount of tokens is no longer available, the function reverts.

As a result, a `buyTokens` call that wants to buy all tokens can fail if any amount of tokens is bought before the call is executed. It may be profitable to intentionally cause reverts of buys, as the auction price decreases with time.

Acknowledged:

Summer.fi has acknowledged the issue and decided not to make any code changes.

9 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

9.1 How to Choose Auction Parameters

Note Version 1

The DutchAuctionLibrary relies on correct usage of the auction parameters to function correctly. The following should be considered when choosing the auction parameters:

1. The auction parameters must be chosen by a trusted role, or default parameters must be used. Allowing arbitrary users to set the auction parameters could lead to loss of funds.
2. The start and end price must be chosen such that the "true market price" is sure to lie between the two prices. However, they should also not be too far apart, as if the price declines by a large amount per block, it can cause lower clearing prices. This is especially important if the auction duration is short.
3. The auction start and end price must be given as "payment token per auction token", taking into account the number of decimals of the tokens. If a token does not support `decimals()`, the default value of 18 is used.
4. If defaults are used, there should be a separate default for every payment/auction token pair, as they will require different start and end prices.
5. In case the relative prices between tokens change significantly, the default auction parameters must be adjusted. Market conditions should be monitored.

9.2 Liquidity Crisis in Aave and Compound

Note Version 1

Users interacting with `AdmiralsQuarters` should be aware that when they want to redeem their aTokens from Aave, by calling `moveFromAaveToAdmiralsQuarters()`, the market might not contain enough funds to fulfill user's withdrawal request. Similarly, redeeming cTokens from Compound, by calling `moveFromCompoundToAdmiralsQuarters()`, could fail.

9.3 Power of Governance

Note Version 1

Even though the governance and other roles are generally trusted to act in favor of the protocol, it is important to note that malicious proposals or governance takeover can still happen.

The governance has the power to (non-exhaustive list):

- grant and revoke roles to arbitrary address
- add arks that can exfiltrate the protocol funds
- exfiltrate protocol funds through auctions
    - ex: exit the liquidity from `PendlePtOracleArk` by using `withdrawExpiredMarket()`

- lock users' funds in the protocol

The keeper has the power to (non-exhaustive list):

- exit the liquidity from `PendlePtOracleArk` by using arbitrary addresses for swapping the fleet asset into the market asset and the other way around

9.4 Sequencer Downtime Can Influence Auction Price

Note **Version 1**

Summer Earn Protocol will be deployed on rollups with centralized sequencers such as Base and Arbitrum One.

As the dutch auction price is dependent on timestamps, the sequencer can influence the price of a dutch auction.

Sequencer downtime could make users temporarily unable to bid in the auction. As a result, the auctioned tokens may be bought later than usual, leading to a clearing price significantly below the true value of the auctioned tokens (or a canceled auction).

In addition to downtime due to technical issues, the sequencer is a trusted role that could act maliciously in the following ways:

- A malicous sequencer could increase the block timestamp to reduce the auction price.
- A malicious sequencer could censor other users' transactions to become the only auction participant. Then they could buy the auctioned tokens at the minimum price.

Performing the malicious actions likely has large external costs for the sequencer, so downtime is much more likely than intentional malicious behavior.

The auction mechanism only works as intended if the sequencer is including bidders' transactions in a fair and timely fashion.

Users should be aware that sequencer downtime can lead to mispriced rewards auctions.

9.5 Staking in `AdmiralsQuarters`

Note **Version 1**

`AdmiralsQuarters.stake()` is meant to stake user's fleet commander shares on his behalf into the rewards manager. However, it assumes that the shares are already in the contract. As `FleetCommander` shares cannot be freely transferred, the `stake()` function is meant to be used in conjunction with `enterFleet()`, with the receiver set to the `AdmiralsQuarters`. Therefore, users should be aware of this behavior and should not call this function separately.

9.6 `AdmiralsQuarters` Uses 0 for Whole Balance

Note **Version 1**

In `AdmiralsQuarters`, zero indicates all the assets when withdrawing tokens, entering the fleet, or exiting the fleet. Similarly, when staking or unstaking, zero indicates all shares. However, in `FleetCommander`, when withdrawing/redeeming, `type(uint256).max` indicates all shares belonging to the user.