SummerFi Security Report

This report was produced for the SummerFi Protocol by Prototech Labs

Mail: info@prototechlabs.dev
Twitter: @Prototech_Labs
Web: www.prototechlabs.dev

Versions:
Final: 18th January 2025

Contents

1. Executive Summary

This report was prepared for the SummerFi team by Prototech Labs, a smart contract consultancy providing security, technical advisory, and code review services. Prototech Labs would like to thank the SummerFi team for giving us the opportunity to review the current state of their protocol.

This document outlines the findings, limitations, and methodology of our review, which is broken down by issue and categorized by severity. It is our hope that this provides valuable findings and insights into the current implementation.

2. Project Overview

This review focused on identifying protocol invariants in order to develop a set of test cases to assess protocol correctness. This then enabled us to execute a test suite against the SummerFi governance contracts, recording and analyzing the results and any deviations from expected behavior, which were then captured as issues.

Project Details:

- **Security Researchers:**
  - Chris Mooney
  - Brian McMichael
  - Derek Flossman
- **Timeline:** throughout 25th Nov 2024 to 17th Jan 2025
- **Code Repository:**
- **https://github.com/OasisDEX/summer-earn-protocol/tree/main/packages/gov-contracts**
- **Commit: 9d679de288a6dc15be796ac91d52773b1270b25f**

3. Introduction

The SummerFi Protocol is a permissionless, passive-earn DeFi product designed to deliver optimized, secure yields while minimizing user effort and diversifying risk. Prototech was tasked with reviewing the Governance contracts, which aim to ensure that the protocol's operations remain safe and transparent. Governance also plays a key role in attracting and compensating key protocol actors by introducing Voting Power Decay, reducing voting power and rewards for inactive participants and their delegators, fostering active engagement and accountability.

In summary, the codebase provides a high level of security and no critical issues were found. It is however worth mentioning that leveraging LayerZero introduces a permissioned actor into the SummerFi protocol, and although this is a known pattern across DeFi, it does add centralization and reliance. This, alongside OpenZeppelin, brings a substantial addition of external functions to the SummerFi token, some of which have been updated since receipt of the initial code commit, and as such we recommend continued test suite analysis as a best practice.

4. Limitations and Report Use

Disclaimer: No assessment can guarantee the absolute safety or security of a software-based system. Further, a system can become unsafe or insecure over time as it and/or its environment evolves. This assessment aimed to discover as many issues and make as many suggestions for improvement as possible within the specified timeframe. Undiscovered issues, even serious ones, may remain. Issues may also exist in components and dependencies not included in the assessment scope.

The software systems herein are emergent technologies and carry with them high levels of technical risk and uncertainty. This report and related analysis of projects to not constitute statements, representations or warranties of Prototech Labs in any respect, including regarding the security of the project, utility of the project, suitability of the project's business model, a project's regulatory or legal status or any other statements, representations or warranties about fitness of the project, including those related to its bug free status. You may not rely on the reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Our complete terms of service can be reviewed here.

Specifically, for the avoidance of doubt, any report published by Prototech Labs does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of any client or project, and is not a guarantee as to the absolute security of any project. Prototech Labs does not owe you any duty by virtue of publishing these reports.

5. Findings Framework

Findings and recommendations are listed in the below section, grouped into broad categories. It is up to the team behind the code to ultimately decide whether the items listed here qualify as issues that need to be fixed, and whether any suggested changes are worth adopting. When a response from the team regarding a finding is available, it is provided.

Findings are given a severity rating based on their likelihood of causing harm in practice and the potential magnitude of their negative impact. Severity is only a rough guideline as to the risk an issue presents, and all issues should be carefully evaluated.

Severity Level Determination		Impact		
		High	Medium	Low
Likelihood	High	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low

Additionally, issues that do not present any quantifiable risk are given a severity of Informational.

6. Findings Overview

Below is an overview of the findings, split by severity, illustrating their status (Fixed/Acknowledged):

7. Critical Risks	**[0]**
7.1 GovernanceRewardsManager.earned() can brick	Fixed

8. High Risks	**[2]**
8.1 SummerVestingWalletFactory: Permissionless creation of vesting wallets can block beneficiaries	Fixed
8.2 SummerToken: _transferVotingUnits is not called, code unused	Fixed

9. Medium Risks	**[3]**
9.1 GovernanceRewardsManager: can stake on behalf of address(0)	Fixed
9.2 SummerGovernor: _cancel is unused	Fixed
9.3 SummerToken: Third party dependency on core token	Acknowledged

10. Low Risks	**[18]**

10.1 If SummerToken.GetPastVotes() requests a timestamp prior to self.originTimestamp then it will panic	Fixed
10.2 package.json: Version lock external dependencies	Fixed
10.3 GovernanceRewardsManager: Reward period can not be modified after initiation	Fixed
10.4 GovernanceRewardsManager: Overflow error on unchecked rewards duration	Fixed
10.5 GovernanceRewardsManager: Unused code/function _initialize	Fixed
10.6 SummerVestingWalletFactory: Unchecked transferFrom in createVestingWallet	Fixed
10.7 getDecayFactor() Requires Explicit Initialization	Fixed
10.8 SummerToken, SummerGovernor, etc.: Avoid tuples in public entry/exit points	Acknowledged
10.9 SummerVestingWalletFactory: token constructor parameter is unchecked	Fixed
10.10 SummerVestingWallet: constructor missing parameter check for `token`	Fixed
10.11 SummerToken: `owner` shadows existing variable in oz/Ownable.sol	Fixed
10.12 SummerGovernor: _proposalThreshold shadows existing variable	Fixed
10.13 GovernanceRewardsManager: _accessManager parameter shadows existing variable	Fixed
10.14 StakingRewardManagerBase: _accessManager shadows existing variable	Fixed
10.15 _rewardTokensList.remove(address(rewardToken)) return value ignored	Fixed
10.16 StakingRewardsManagerBase: _rewardTokensList.add(address(rewardToken) ignores return value	Fixed
10.17 SummerVestingWalletFactory: Use safeTransferFrom Instead of transferFrom for Enhanced Safety and Compatibility	Fixed
10.18 SummerVestingWallet: recallUnvestedTokens Ignores Return Value of IERC20(token).transfer for Unvested Performance Tokens	Fixed
10.19 stakeOnBehalfOf address(0)	Acknowledged

10.20 notifyRewardAmount divide by zero	Acknowledged
10.21GovernanceRewardsManager: Consider scoping rewardsDuration to reasonable amounts	Acknowledged

11. Informational Findings	**[13]**
11.1 GovernanceRewardsManager: Missing accessors for valid rewards tokens	Fixed
11.2 Inconsistent Datatype parameters between SummerToken and SummerGovernor	Fixed
11.3 SummerToken: Consider a `view` function to obtain a delegatee's vote power with calculated decay	Fixed
11.4 StakingRewardsManagerBase has no function to assist iteration through reward tokens	Fixed
11.5 SummerGovernor: Provide external view function to determine whether account has been initialized	Acknowledged
11.6 StakingRewardsManagerBase: Consider making stakingToken immutable	Fixed
11.7 StakingRewardsManagerBase: Avoid type enforcement on parameters and return values	Fixed
11.8Undocumented Loss of Voting Power at Maximum Delegation Depth	Fixed
11.9 SummerToken: `rewardsManager` and `vestingWalletFactory` can be immutable	Fixed
11.10 ProtocolAccessManaged: _accessManager can be immutable	Fixed
11.11 ISummerToken does not include all the interfaces from SummerToken	Acknowledged
11.12 GovernanceRewardsManager: unstakeOnBehalfOf doc nit	Acknowledged
11.13 General Compiler warnings	Acknowledged
11.14 WrappedSummerToken: Hardcoded values in name and symbol	Acknowledged

12. Gas Optimization	**[4]**
12.1 SummerVestingWallet: timeBasedVestingAmount can be immutable	Acknowledged

12.2 SummerVestingWallet: Gas savings on loops	Acknowledged
12.3 SummerToken: Unnecessary import of IGovernanceRewardsManager	Fixed
12.4 gov-contracts: gas report	Acknowledged

7. Critical Risks

7.1 GovernanceRewardsManager.earned() can brick

Context: packages/rewards-contracts/src/contracts/StakingRewardsManagerBase.sol#L318-L323

Description: On the following sequence there is an underflow because of a `0 - 21` in the numerator.

The sequence:

```
Unset
function
test_regression_invariant_SEP_rewardBalanceCoversEarned_ccfbadac_
failure() external {
        _setMaxLeap(2400);

_summerTokenHandler.permit(97550486811007272872446395071006013790
07041278481175684700299090696236283461, 309388254228629941,
312929571003046511919280,
25833583690962132226056259339044448715356614141146671228412441,
43539073856578365375958520274269600248733344213951983792168809556
6152784);

_summerTokenHandler.addToWhitelist(12908375424644438317323442768 7
03814713700736326, 85717502507952425947578863522339949 45);
        _governanceRewardsManagerHandler.stakeOnBehalfOf(24450,
849521988, 8417);

_governanceRewardsManagerHandler.notifyRewardAmount(52107, 17990,
21155, 1978);
```

```
_governanceRewardsManagerHandler.getRewardForByAddress(1638637330
180918789784039494703546085793935356600408897);
        _summerTokenHandler.delegate(166742040047670150469555,
632187652645067433701441370327072172169029703244781943039191679661
61675384);

_governanceRewardsManagerHandler.stakeOnBehalfOf(4518740217265209
, 491735,
129339218229999075142587636710380193265896286059748832100217289543
66030188);

_governanceRewardsManagerHandler.removeRewardToken(385987712,
2655);

        invariant_SEP_rewardBalanceCoversEarned();
    }
```

The code section with console logging:

Unset

```
    /*
     * @notice Internal function to calculate earned rewards for
an account
     * @param account The address to calculate earnings for
     * @param rewardToken The reward token to calculate earnings
for
     * @return The amount of reward tokens earned
     */
    function _earned(
        address account,
        IERC20 rewardToken
    ) internal view returns (uint256) {
        console.log("_balances[account]", _balances[account]);
```

```
        console.log("rewardPerToken",
rewardPerToken(rewardToken));
        console.log("userRewardPerTokenPaid",
userRewardPerTokenPaid[rewardToken][account]);
        console.log("rewards", rewards[rewardToken][account]);
        console.log("account", account);
        console.log("rewardToken", address(rewardToken));

        return
            (_balances[account] *
                (rewardPerToken(rewardToken) -

userRewardPerTokenPaid[rewardToken][account])) /
            Constants.WAD +
            rewards[rewardToken][account];
    }
```

When attempting to compute the invariant we get the `earned()` value, which reverts for this account:

Unset

```
    ├─ [11585] GovernanceRewardsManager::earned(Actor:
[0x65853931aCE16AaB0D5b5EAf44F3cdd609C7b099], SummerToken:
[0x212224D2F2d262cd093eE13240ca4873fcCBbA3C]) [staticcall]
    |   ├─ [997] SummerToken::delegates(Actor:
[0x65853931aCE16AaB0D5b5EAf44F3cdd609C7b099]) [staticcall]
    |   |   └─ ← [Return] Actor:
[0x65853931aCE16AaB0D5b5EAf44F3cdd609C7b099]
    |   ├─ [0] console::log("_balances[account]", 0) [staticcall]
    |   |   └─ ← [Stop]
    |   ├─ [0] console::log("rewardPerToken", 0) [staticcall]
    |   |   └─ ← [Stop]
    |   ├─ [0] console::log("userRewardPerTokenPaid", 21)
[staticcall]
    |   |   └─ ← [Stop]
```

```
    |   ├─ [0] console::log("rewards", 0) [staticcall]
    |   |   └─ ← [Stop]
    |   ├─ [0] console::log("account", Actor:
[0x65853931aCE16AaB0D5b5EAf44F3cdd609C7b099]) [staticcall]
    |   |   └─ ← [Stop]
    |   ├─ [0] console::log("rewardToken", SummerToken:
[0x212224D2F2d262cd093eE13240ca4873fcCBbA3C]) [staticcall]
    |   |   └─ ← [Stop]
    |   └─ ← [Revert] panic: arithmetic underflow or overflow
(0x11)
    └─ ← [Revert] panic: arithmetic underflow or overflow (0x11)
```

SummerFi: Fixed

8. High Risks

8.1 SummerVestingWalletFactory: Permissionless creation of vesting wallets can block beneficiaries

Context: SummerVestingWalletFactory.sol#L18-L23

Description:
The VestingWallet creation process is permissionless in this contract, allowing any entity to access the contract directly after it is deployed by the `SummerToken`, create a new vesting wallet with a specific `beneficiary`, and designate themselves as the wallet's admin. This action blocks the creation of any new vesting wallets for that particular `beneficiary` address.

Here, `beneficiary` is an externally-supplied parameter, and the caller of the function becomes the wallet's `Guardian`. Any end user can create new vesting wallets for unclaimed beneficiaries, which adds those beneficiaries to the tracked list in the Factory and prevents legitimate wallets for the same beneficiaries from being created.

Recommendation:
Implement a validation mechanism that ensures only authorized entities can create new

vesting wallets or introduce logic to prevent malicious actors from blocking legitimate wallet creation for specific beneficiaries.

SummerFi:
- Commit:
https://github.com/OasisDEX/summer-earn-protocol/pull/193/commits/6559e21fef88581f569309bdf266c0de659f6c22
- Using central protocol access alongside a new FOUNDATION_ROLE

8.2 SummerToken: _transferVotingUnits is not called, code unused

Context:
SummerToken.sol#L346-L373
SummerToken.sol#L398

Description:
The `_transferVotingUnits` function code is defined locally but is not utilized. The `_handleVestingWalletVotingTransfer` function calls `super._transferVotingUnits()`, which propagates the call to the parent contract rather than using the locally overridden function.
This creates a potential issue where the intended behavior of the overridden function in the local contract is bypassed. Tests are needed to ensure the correct behavior is maintained.

Recommendation:

- Do not use the `super` keyword if the intended behavior is to invoke the locally overridden `_transferVotingUnits` function.
- Update the `_handleVestingWalletVotingTransfer` function to call the local `_transferVotingUnits` implementation.
- Add comprehensive tests to verify the functionality and ensure the intended behavior is correctly executed.

SummerFi:
- This is intended behaviour. _transferVotingUnits (in SummerToken.sol) overrides _transferVotingUnits (in Votes.sol). `_handleVestingWalletVotingTransfer is part of the implementation of our` _transferVotingUnits override so if

_handleVestingWalletVotingTransfer were to call _transferVotingUnits instead of super._transferVotingUnits we'd end up in an infinite recursion.

9. Medium Risks

9.1 GovernanceRewardsManager: can stake on behalf of address(0)

Context: GovernanceRewardsManager.sol#L94

Description:
Users can `stakeOnBehalfOf` where `receiver` is `address(0)`. Since `unstakeOnBehalfOf` is unimplemented, this appears to lock the funds on behalf of `address(0)`.

Users may inadvertently lock funds in `address(0)` by passing it as the receiver param either inadvertently with a malformed signature, or intentionally thinking they are providing a neutral address value.

This test can be used with `./packages/rewards-contracts/test/StakingRewardsManagerBase.t.sol`

Recommendation: Add an guard against `address(0)` for the `receiver` in `stakeOnBehalfOf`

SummerFi:
- Commit: https://github.com/OasisDEX/summer-earn-protocol/pull/228/commits/b4e82571eb238c23dda25862b5e1e30571d3002e

9.2 SummerGovernor: _cancel is unused

Context: SummerGovernor.sol#L527-L548

Description:
The `_cancel` internal function is defined in the `SummerGovernor` contract but is not

used within the implementation. The `cancel` function instead calls `super.cancel()`, which bypasses the overridden `_cancel` internal function and uses the inherited implementation instead.
This may indicate an unintentional design flaw if `_cancel` was meant to be invoked as part of the `cancel` function's logic.

Recommendation:

- Call `_cancel` directly from the `cancel` function if the local implementation is intended to be used.
- Avoid relying on `super.cancel()` if it overrides the expected local behavior.
- Add thorough tests to verify the correctness and intended functionality of the `cancel` process.

SummerFi:
- Commit:
https://github.com/OasisDEX/summer-earn-protocol/pull/193/commits/9164b738589a919f6ebc5a5d34a11805410db78f

9.3 SummerToken: Third party dependency on core token

Context: SummerToken.sol#L5

Description:
The core governance token depends on the LayerZero protocol for bridging functionality, introducing significant complexity and reliance on a single third-party provider. This dependency creates several potential risks:

- Failure or insolvency of LayerZero could disrupt the bridging feature and limit the token's usability.
- Delays or refusal by LayerZero to support new chains could restrict SummerFi's ability to capitalize on growth opportunities on emerging chains.

Discussions with the client suggest that governance activities will primarily occur on the Base chain, making the necessity of this feature on bridged chains less critical.

Recommendation:

- Assess whether LayerZero's bridging functionality is essential for the token's intended use cases.
- If governance activities will remain on the primary chain, consider managing governance on secondary chains via tokens sent using the respective chain's native bridge.
- Evaluate the trade-off between the added complexity and interdependence of the OFT library and the actual utility it provides for the protocol.
- If LayerZero is retained, document its criticality and ensure contingency plans are in place for potential failures or limitations of the third-party service.

SummerFi: Acknowledged

10. Low Risks

10.1 If SummerToken.GetPastVotes() requests a timestamp prior to self.originTimestamp then it will panic

Context: GovernanceRewardsManager.sol

Description:

If `SummerToken.GetPastVotes()` requests a timestamp prior to `self.originTimestamp` then this library function will revert with `[Revert] panic: arithmetic underflow or overflow (0x11)`.

```
Unset
   ├─ [12258] SummerToken::getPastVotes(@Actor:
[0x8C5411ac3BA9345B70F42f575178aA177E6824Cd], 1663224161
[1.663e9]) [staticcall]
    │   └─ ← [Revert] panic: arithmetic underflow or
overflow (0x11)
    └─ ← [Revert] panic: arithmetic underflow or overflow
(0x11)
```

Recommendation:

This is because `timestamp < self.originTimestamp` from the `initialization()`. One recommendation would be to make this change to the library.

```
Unset
diff --git
a/packages/voting-decay/src/VotingDecayLibrary.sol
b/packages/voting-decay/src/VotingDecayLibrary.sol
index 0eca8942..09b7ea33 100644
--- a/packages/voting-decay/src/VotingDecayLibrary.sol
+++ b/packages/voting-decay/src/VotingDecayLibrary.sol
@@ -524,6 +524,10 @@ library VotingDecayLibrary {
         address accountAddress,
         uint256 timestamp
     ) internal view returns (uint256) {
+        if (timestamp < self.originTimestamp) {
+            return 0; // Or WAD, depending on desired
behavior
+        }
         uint224 checkpointValue = self
             .decayFactorCheckpoints[accountAddress]
             .upperLookup(uint32(timestamp));
```

We do, however, recommend checking the code for anywhere `getPastVotes()` may be called in critical paths to ensure it doesn't trigger this panic.

SummerFi:
- Commit:
https://github.com/OasisDEX/summer-earn-protocol/commit/394fe915924b316053da4ebd77ad816357658f29
- Addressed in a separate branch (+ commit) from audit fixes branch

10.2 package.json: Version lock external dependencies

Context: package.json

Description:

External dependencies are not version-locked. External dependencies can be updated inadvertently and subject this code to a supply-chain attack.

Recommendation:

Lock all external dependencies to a specific version. If a new version of the external dependency is required prior to launch, update it manually and test all code against that version.

SummerFi:
Commit:
https://github.com/OasisDEX/summer-earn-protocol/pull/228/commits/8b59499ad99c45bc07dce5e48af63dabfda2cd58

10.3 GovernanceRewardsManager: Reward period can not be modified after initiation

Context: StakingRewardsManagerBase.sol#L231-L242

Description:

The rewards duration for a specific token can not be modified after notification. A reward token that has been initiated for too long will effectively lock that token from being eligible for rewards afterward.

See the following test added to `GovernanceRewardsManager.t.sol`

```
Unset
function test_setRewardsDurationAfterNotify() public {
        uint256 rewardAmount = 1000 * 1e18;

        vm.prank(address(mockGovernor));
        stakingRewardsManager.notifyRewardAmount(
            IERC20(address(rewardTokens[0])),
            rewardAmount,
```

```
            90001 days  // Whoops, too long
        );

        vm.warp(block.timestamp + 1 days);

        vm.prank(address(mockGovernor));
        stakingRewardsManager.setRewardsDuration(
            IERC20(address(rewardTokens[0])),
            7 days
         );
    }
```

Recommendation:

Consider allowing for some method to recover from a rewards period that is too long or limit the maximum reward period to prevent locking tokens from the rewards structure. Governance actions may be subject to human error here and there should be a method to recover from this. `removeRewardsToken` also has a similar guard preventing removal of a rewards token when the duration is not complete.

SummerFi:
Commit:
https://github.com/OasisDEX/summer-earn-protocol/pull/228/commits/cb34260826db813bee84fba4626fd7924af9fbc2

10.4 GovernanceRewardsManager: Overflow error on unchecked rewards duration

Context: StakingRewardsManagerBase.sol#L225-L227

Description:

In `notifyRewardAmount`, the function accepts an arbitrary `uint256` as the rewardsDuration, which can overflow when added to `block.timestamp`. The duration

itself is stored as a `uint256`, which could lead to otherwise impractical durations being set, and if done so accidentally, can affect the rewards schedule for that particular `rewardsToken` permanently, as the duration can not be changed later due to the `CannotChangeRewardsDuration` check.

Recommendation:

Consider at least requiring that supplied durations are `< type(uint48).max - block.timestamp` or a more conservative bound, and issue a custom error in the event a large duration is supplied.

SummerFi:

Also covered by this commit:
https://github.com/OasisDEX/summer-earn-protocol/pull/228/commits/cb34260826db813bee84fba4626fd7924af9fbc2
In practice any duration sufficient to trigger an overflow will also be caught by this error.

10.5 GovernanceRewardsManager: Unused code/function _initialize

Context: GovernanceRewardsManager.sol#L76-L83

Description:
The `GovernanceRewardsManager._initialize()` function is internal and appears to be unused elsewhere in the contract.
Tests pass successfully even when the function is removed, suggesting it is untested.
The function seems to be intended to populate the `stakingToken` in the `StakingRewardsManagerBase`. However, this is already handled directly in the constructor, and `_initialize` is not invoked elsewhere. As a result, the event emitted within this function is never fired.

Recommendation:

- Ensure `_initialize()` is called appropriately and add tests to verify the initialization process and event.
- Alternatively, consider 11.6 and remove the initialize function entirely.

SummerFi:
Commit:
https://github.com/OasisDEX/summer-earn-protocol/commit/2815b76ebe69e8569490dcdc30e991ae11a50287

10.6 SummerVestingWalletFactory: Unchecked transferFrom in createVestingWallet

Context: SummerVestingWalletFactory.sol#L53

Description:
The `transferFrom()` call in the `createVestingWallet` function is not checked for successful completion. Additionally, the constructor accepts an unchecked and arbitrary `token` parameter, which introduces the potential for misuse.
While this may not pose an issue if the contract is exclusively used to manage the predefined `SummerToken`, it may create vulnerabilities if the contract is extended for other use cases in the future.

Recommendation:

- Add a sanity check to ensure `transferFrom()` completes successfully, such as by verifying the returned value or checking balances post-transfer.
- Evaluate the potential for reentrancy attacks and consider adding reentrancy guards if applicable.
- If this contract is intended solely for managing `SummerToken`, document this limitation clearly to avoid future misuse.
- If broader use is intended, implement stricter validation of the `token` parameter and ensure robust checks are in place.

SummerFi:
Commit:
https://github.com/OasisDEX/summer-earn-protocol/pull/193/commits/d257f1796ddd112318a1c155f02a042fb9caf0e4

10.7 getDecayFactor() Requires Explicit Initialization

Context: VotingDecayLibrary.sol

Description:

The current implementation requires every account to have `updateDecayFactor()` called before `getDecayFactor()` can be used, even though uninitialized accounts effectively have a decay factor of WAD (1e18). This creates unnecessary friction and gas costs for users. What's more, since `getDecayFactor()` is called in a number of locations like `getVotingPower()` and `earned()`, and any of those functions may be called by third parties, this side-steps an entire set of potentially defective terminal states that could result in bricked integrations. This escalates the issue from informational to low.

The issue stems from `_getDecayFactorWithDepth()` reverting with `AccountNotInitialized()` if an account hasn't been initialized, rather than returning the default WAD value:

```
Unset
if (!_hasDecayInfo(self, accountAddress)) {
    revert AccountNotInitialized();
}
```

This can be reproduced by attempting to call `getDecayFactor()` on any new account that hasn't had `updateDecayFactor()` called on it first.

Recommendation:

There are two potential approaches to handle uninitialized accounts:

1. Return WAD for uninitialized accounts:

```
Unset
if (!_hasDecayInfo(self, accountAddress)) {
    return WAD;  // Return default value instead of
reverting
}
```

This would:

- Eliminate unnecessary gas costs from initialization transactions
- Improve UX by removing a required setup step
- Maintain the same mathematical correctness since WAD is the starting value anyway
- Avoid potentially defective terminal states

However, this approach has a significant trade-off: It would allow accounts to maintain full voting power indefinitely without ever initializing, which could defeat the purpose of the decay mechanism.

2. Apply full decay to uninitialized accounts (our recommendation):

```
Unset
if (!_hasDecayInfo(self, accountAddress)) {
    return _calculateDecayFactor(
        WAD,
        block.timestamp - transferEnableDate, // Use
contract deployment as start
        self.decayRatePerSecond,
        self.decayFreeWindow,
        self.decayFunction
    );
}
```

This would:

- Eliminate unnecessary gas costs from initialization transactions
- Improve UX by removing a required setup step
- Maintain the decay mechanism's purpose by ensuring all accounts decay
- Avoid potentially defective terminal states

Additionally, to help integrations detect uninitialized accounts before calling functions that might revert, consider adding a public view function:

```
Unset
function isAccountInitialized(address account) external
view returns (bool) {
    return
decayState.decayInfoByAccount[account].lastUpdateTimestamp
!= 0;
}
```

This would allow third parties to check initialization status before attempting operations that depend on `getDecayFactor()`.

SummerFi:
Commit:
https://github.com/OasisDEX/summer-earn-protocol/pull/193/commits/4b27aeeb4916489f0a45b51486e9d26436e97620

10.8 SummerToken, SummerGovernor, etc.: Avoid tuples in public entry/exit points

Context:
SummerToken.sol#L59-L61
SummerGovernor.sol#L81-L83

Description:
The use of struct tuples in exposed parameters and return values for constructors and functions adds unnecessary complexity for interaction and analysis. Interacting with these structs requires the importation of the struct definition and additional parsing for analysis, creating friction for developers, users, and data analysts.
Struct tuples are particularly problematic when they include dynamic types like `bytes` or `string`, which can lead to compiler-related bugs, such as the tuple re-encoding head overflow issue described here.

Recommendation:

- Avoid using struct tuples in exposed parameters and return values for constructors and external/public functions.

- Use discrete parameter inputs for these functions to simplify user interaction and analysis.
- Review and refactor existing code to remove unnecessary struct tuples, ensuring compatibility and robustness.
- Add tests to verify the functionality remains intact after refactoring.

SummerFi: Acknowledged

10.9 SummerVestingWalletFactory: token constructor parameter is unchecked

Context: SummerVestingWalletFactory.sol#L15

Description:
The `token` parameter in the constructor is unchecked, allowing for the possibility of an unintentional `address(0)` being passed. This could result in unexpected behavior or errors when the contract interacts with the token.

Recommendation:

- Add a validation check to ensure that the `token` parameter is not set to `address(0)`.
- Include tests to verify the validation logic and ensure the contract behaves as expected when invalid addresses are passed.

SummerFi:
Commit:
https://github.com/OasisDEX/summer-earn-protocol/commit/d257f1796ddd112318a1c155f02a042fb9caf0e4

10.10 SummerVestingWallet: constructor missing parameter check for token

Context: SummerVestingWallet.sol#L78

Description:
The `token` parameter is used without validation, which could allow `address(0)` to be

passed inadvertently. Using `address(0)` as the token address could lead to unexpected behavior or errors in contract operations.

Recommendation:

- Add a check to ensure the `token` parameter is not set to `address(0)`.
- Include tests to verify that the contract properly rejects invalid token addresses and functions as expected with valid inputs.

SummerFi:
Commit:
https://github.com/OasisDEX/summer-earn-protocol/commit/d257f1796ddd112318a1c155f02a042fb9caf0e4

10.11 SummerToken: owner shadows existing variable in oz/Ownable.sol

Context: SummerToken.sol#L171

Description:
The `owner` parameter in this context shadows an existing variable that is inherited from the `openzeppelin/Ownable` contract. Variable shadowing can lead to confusion and potential bugs, as it may obscure the distinction between the local parameter and the inherited state variable.

Recommendation:

- Rename the `owner` parameter to avoid shadowing the inherited `owner` variable.
- Use a more descriptive name, such as `initialOwner` or `newOwner`, to clarify its purpose.
- Review and update any references to this parameter throughout the contract.

SummerFi:
Commit:
https://github.com/OasisDEX/summer-earn-protocol/commit/77d5f26b32e104bfb856ab51cef0ecab2b8fbb1e

10.12 SummerGovernor: _proposalThreshold shadows existing variable

Context: SummerGovernor.sol#L479-L480

Description:
The `_proposalThreshold` parameter in this context shadows an internal variable inherited from the `openzeppelin/GovernorSettings` contract. Variable shadowing can lead to confusion and unintended behavior, especially when interacting with inherited functionality.

Recommendation:

- Rename the `_proposalThreshold` parameter to avoid shadowing the inherited internal variable.
- Use a more descriptive and unique name, such as `initialProposalThreshold` or `constructorProposalThreshold`, to clearly differentiate it from the inherited variable.
- Review and update all references to this parameter in the constructor and elsewhere in the contract as needed.

SummerFi:
Commit:
https://github.com/OasisDEX/summer-earn-protocol/pull/193/commits/48222857def9e3122d28775c616ea306e301c951

10.13 GovernanceRewardsManager: _accessManager parameter shadows existing variable

Context: GovernanceRewardsManager.sol#L68-L71

Description:
The `_accessManager` parameter in this context shadows an internal variable inherited from `StakingRewardsManagerBase` via `ProtocolAccessManaged.sol`. Variable shadowing can lead to confusion and unintended behavior when interacting with both the local and inherited variables.

Recommendation:

- Rename the `_accessManager` parameter to avoid shadowing the inherited variable.
- Use a distinct and descriptive name, such as `constructorAccessManager` or `inputAccessManager`, to clarify its purpose and distinguish it from the parent contract's variable.
- Update all references to this parameter within the constructor and any other related functionality in the contract.

SummerFi:
Commit:
https://github.com/OasisDEX/summer-earn-protocol/pull/193/commits/77d5f26b32e104bfb856ab51cef0ecab2b8fbb1e

10.14 StakingRewardManagerBase: _accessManager shadows existing variable

Context: StakingRewardsManagerBase.sol#L77

Description:
The `_accessManager` parameter in this context shadows an internal variable inherited from `ProtocolAccessManaged.sol`, which the `StakingRewardsManagerBase` contract inherits. Shadowing inherited variables can lead to confusion and potential errors in functionality when interacting with both the local parameter and the inherited variable.

Recommendation:

- Rename the `_accessManager` parameter to avoid shadowing the inherited variable.
- Use a distinct and descriptive name, such as `constructorAccessManager` or `inputAccessManager`, to differentiate it from the parent contract's variable.
- Update all references to this parameter within the constructor and any related functionality in the contract.

SummerFi:
Commit:
https://github.com/OasisDEX/summer-earn-protocol/pull/193/commits/df25f7417c0f141685e083d720a607af5c9e2a52

10.15 _rewardTokensList.remove(address(rewardToken)) return value ignored

Context: StakingRewardsManagerBase.sol#L265

Description:
The operation at this line will return `true` if the element was successfully removed and `false` if it was not present in the collection. Currently, the return value is not checked, which could lead to the logic proceeding with incorrect assumptions, potentially resulting in unexpected behavior or silent failures.

Recommendation:

- Add a `require()` statement or equivalent check to ensure the removal was successful before proceeding with subsequent logic.
- Include test cases to validate scenarios where the removal fails, ensuring the contract handles such cases appropriately and robustly.

SummerFi:
Commit:
https://github.com/OasisDEX/summer-earn-protocol/pull/228/commits/cb34260826db813bee84fba4626fd7924af9fbc2

10.16 StakingRewardsManagerBase: _rewardTokensList.add(address(rewardToken) ignores return value

Context: StakingRewardsManagerBase.sol#L193

Description:
The `add()` function used here returns a boolean value: `true` if the element was successfully added and `false` if it already exists in the collection. Currently, the return value is not checked, which could allow the contract logic to proceed without ensuring that the addition was successful.

Recommendation:

- Add a `require()` statement or explicit check around the `add()` function call to verify that the operation succeeded.
- Include tests to validate scenarios where the addition fails, ensuring the contract handles such cases appropriately.

SummerFi:
Commit:
https://github.com/OasisDEX/summer-earn-protocol/pull/228/commits/cb34260826db813bee84fba4626fd7924af9fbc2

10.17 SummerVestingWalletFactory: Use safeTransferFrom Instead of transferFrom for Enhanced Safety and Compatibility

Context: SummerVestingWalletFactory.sol#L53

Description:
The `transferFrom` function is used at this location for token transfers. While functional, using `transferFrom` does not include additional safety checks provided by `safeTransferFrom`, which ensures that the receiving contract can handle the token and mitigates the risk of tokens being sent to incompatible contracts.

Recommendation:

- Replace `transferFrom` with `safeTransferFrom` to ensure compatibility and enhance safety during token transfers.
- Verify that the IERC20 contract implementation supports safeTransferFrom. Add or import SafeERC20 from OpenZeppelin if not already included.
- Add tests to confirm correct behavior when using safeTransferFrom, especially for edge cases like transferring to non-compliant contracts.

SummerFi:
Commit:
https://github.com/OasisDEX/summer-earn-protocol/commit/bd30818769e3f1bbb1e7003223806a4e274dab43

10.18 SummerVestingWallet: recallUnvestedTokens Ignores Return Value of IERC20(token).transfer for Unvested Performance Tokens

Context: SummerVestingWallet.sol#L124

Description:
The `transfer()` function is used at this location for token transfers, but its return value is ignored. This could lead to silent failures if the transfer fails. Using `safeTransfer()` from OpenZeppelin's `SafeERC20` library ensures the transfer is successful by reverting the transaction in case of failure.

Recommendation:

- Replace `transfer()` with `safeTransfer()` to ensure successful execution of the transfer and mitigate risks of silent failures.
- Import OpenZeppelin's SafeERC20 library if not already included.
- Update tests to confirm the behavior of safeTransfer() under various scenarios, such as insufficient balances or invalid recipients.

SummerFi:
Commit:
https://github.com/OasisDEX/summer-earn-protocol/pull/193/commits/bd30818769e3f1bbb1e7003223806a4e274dab43

10.19 StakeOnBehalfOf address(0)

Context: GovernanceRewardsManager.stakeOnBehalfOf does not revert when the receiver is the 0 address.

Description: This allows tokens to be added to the GovernanceRewardsManager on behalf of address(0), effectively burning them, since there is no way to unlock them.

Recommendation: To fix, you'd need to add something like:

if (receiver == address(0)) { revert CannotStakeZeroAddress(); }

This creates a balance invariant because `address(0)` cannot hold a balance in the stakingToken.

Example:

```
Unset
function test_regression_invariant_ST_balanceSum_2e78768b_failure() external {
        _setMaxLeap(2400);
        _summerTokenHandler.approve(14485, 16999,
228407828082926521985578023698233863965214214020108421782211392588344461466645);

_summerTokenHandler.enableTransfers(32236086649843664542764775387999073316142590986760826148692504077579495159 2);

_governanceRewardsManagerHandler.stakeOnBehalfOf(142230531980466670342374290100731173222484988686421408178645569, 117360633980989770961122,
171816983516785747495421915506542 2);

        invariant_ST_balanceSum();
    }
```

SummerFi: Acknowledged

10.20 notifyRewardAmount divide by zero

Context: StakingRewardsManagerBase._notifyRewardAmount() when a token that has already been added has setRewardsToken set to 0 after the initial period. We get a divideByZero error when attempting to add a new rewards period.
Description: With this unit test you can add to GovernanceRewardsManager.calculations.t.sol to explore this: https://gist.github.com/brianmcmichael/cd98f1a999d90b616e9219cfcb57f63a

Note that setting the rewardsDuration to 0 via the setter will effectively disable new rewards periods from being added until setRewardsDuration() is invoked again with a value > 0.

```
Unset
```

```
function test_regression_invariant_ST_permit_b4a1c71f_failure() external {
        _setMaxLeap(2400);
        _governanceRewardsManagerHandler.notifyRewardAmount(11749, 23137, 4306,
153);

_governanceRewardsManagerHandler.setRewardsDuration(63828681538402403430399977926365589618640804784924996870426826046859371784 82,
98929628814680382364998214083426140574885917485797657687563121735279169452427,
0);
        _governanceRewardsManagerHandler.notifyRewardAmount(0,
1081908549853678848266967167247198652, 1, 127753576791540);

        invariant_ST_permit();
    }
```

SummerFi: Acknowledged

10.21 GovernanceRewardsManager: Consider scoping rewardsDuration to reasonable amounts

Context: StakingRewardsManagerBase.sol#L372

Description: The staking rewards allows a new rewards duration to be an impractically large number, up to max uint, which can create practical issues when a reward duration is too long because the period for a given token can not be adjusted during it's duration. A faulty input here can effectively block new rewards from being issued for a token if inadvertently set to a large uint duration.

The `setRewardsDuration` will not allow this to be corrected if the period has not ended, so a faulty parameter here can lock new rewards indefinitely.

Recommendation: Recommend adding a check that the newRewardsDuration is less than some amount of time that a maximum rewards period can last.

`if (newRewardsDuration > 5 years) { revert RewardsDurationTooLong() };

Or whatever the max period would be expected.

Also consider whether this value can be fixed with `setRewardsDuration` via a governance action to correct any issues caused by faulty parameters.

SummerFi: Acknowledged

11. Informational Findings

11.1 GovernanceRewardsManager: Missing accessors for valid rewards tokens

Context: GovernanceRewardsManager.sol

Description:

The GovernanceRewardsManager includes `reverts` in many places when `RewardTokenDoesNotExist`, however, the `rewardsTokenList` is an internal structure and there are no direct accessors to determine whether a token is in the valid list or not.

Recommendation:

Add an external `function isRewardToken(address) external view returns (bool)` to allow an integrator to determine whether a rewardToken is valid prior to making a `getReward(address)` call that will result in a revert.

Alternatively, add an external `_rewardToken.length` accessor function so that `rewardTokens(index)` can be iterated over by integrators. See 11.4

SummerFi:
Commit:
https://github.com/OasisDEX/summer-earn-protocol/pull/228/commits/b483ec7d3aa6cb887347c66f023f575bd3147784

11.2 Inconsistent Datatype parameters between SummerToken and SummerGovernor

Context: SummerToken.sol, SummerGovernor.sol

Description:

The ISummerToken and ISummerGovernor `TokenParams` and `GovernorParams` have inconsistent datatypes.

In `TokenParams`: `accessManager` is an `address` datatype
in `GovernorParams`: `accessManager` requires an `IProtocolAccessManager` datatype

Recommendation:

Whenever possible, use native datatypes for parameters and returns. In this case, recommend using `address` datatype for `accessManager` and removing the `IProtocolAccessManager` import from this interface.

SummerFi:
Commit:
https://github.com/OasisDEX/summer-earn-protocol/pull/228/commits/cc10298d94598724010bd697977da59461ab3466

11.3 SummerToken: Consider a view function to obtain a delegatee's vote power with calculated decay

Context: SummerToken.sol

Description:

The voting power of a delegatee is calculated internally in the VotingDecayLibrary using imported PRBMath in VotingDecayMath. This involves rounding and precision control that is difficult to accurately replicate without rebuilding the libraries. Simply applying the VotingDecayFactor to the balance results in values that are different from the result produced internally.

Recommendation:

Add an external `view` function that returns the current voting power of a delegatee with the decay factor applied to their balance with a result equal to the internally calculated value.

SummerFi:
Commit:
https://github.com/OasisDEX/summer-earn-protocol/pull/228/commits/3158410e9002fffa15126f51b3effa300b569062

11.4 StakingRewardsManagerBase has no function to assist iteration through reward tokens

Context: StakingRewardsManagerBase.sol#L83-L89

Description:
The `StakingRewardsManagerBase` contract exposes a `rewardTokens(uint256 index)` function to iterate through reward tokens, but does not provide a way to determine the total number of reward tokens. This forces users to either:

- Blindly increment the index until they hit an IndexOutOfBounds revert
- Try to track the number of tokens externally by monitoring events

Both approaches are error-prone and gas inefficient. The contract uses an EnumerableSet internally which already tracks the length, but this information is not exposed publicly.
This makes it difficult for integrators to safely interact with all reward tokens and could lead to incomplete reward processing if tokens are added/removed between operations.

Recommendation:
Add a public view function to expose the reward tokens length:

```
Unset
function rewardTokensLength() external view returns (uint256) {
    return rewardTokensList.length();
}
```

This allows safe iteration:

```
Unset
uint256 numTokens = rewardsManager.rewardTokensLength();
for(uint256 i = 0; i < numTokens; i++) {
    IERC20 token = rewardsManager.rewardTokens(i);
    // Process token
}
```

SummerFi:
Commit:
https://github.com/OasisDEX/summer-earn-protocol/pull/228/commits/61385fa67494ca20b19edef611ae57a53548be19

11.5 SummerGovernor: Provide external view function to determine whether account has been initialized

Context: SummerGovernor.sol:VotingDecayLibrary.sol

Description:

There does not appear to be a `view`able way to determine if an account has been activated. There is an internal function `_hasDecayInfo` that is used in the SummerGovernor that is used to determine if an account has been initialized, and which will revert if the account has no decay info.

Recommendation:

Provide an `external view` function that can be used to determine if a prospective account has been activated.

SummerFi: Acknowledged

11.6 StakingRewardsManagerBase: Consider making stakingToken immutable

Context: StakingRewardsManagerBase.sol#L45

Description:

The `stakingToken` in `StakingRewardsManagerBase` is set via an `initialize` function and does not have any functions to change this token after it is set. This can be made immutable with a few modifications.

Recommendation:

- set `stakingToken` to be an `immutable` value. Use `address` as the external type here (See 11.7)
- remove the `initialize()` function as this is internal and presumed to be set in the `GovernanceRewardsManager` constructor.
- add `stakingToken` to the `StakingRewardsManagerBase` and set the immutable value in the constructor.

SummerFi:
Commit:
https://github.com/OasisDEX/summer-earn-protocol/pull/228/commits/0d6e8539b686002ee3378c1d95a53ab61bddaba9

11.7 StakingRewardsManagerBase: Avoid type enforcement on parameters and return values

Context:

StakingRewardsManagerBase.sol#L85 ;) external view override returns (IERC20) {
StakingRewardsManagerBase.sol#L98 ; IERC20 rewardToken
StakingRewardsManagerBase.sol#L107 ; function rewardPerToken(IERC20 rewardToken) public view returns (uint256) {
StakingRewardsManagerBase.sol#L122 ; IERC20 rewardToken
StakingRewardsManagerBase.sol#L184 ; IERC20 rewardToken,

Description:

In the StakingRewardsManagerBase, many of the functions require a type declaration for parameters and returns. These don't add any validations in and of themselves that would guarantee that the passed address is actually IERC20 compliant, but forces integrations to import an IERC20 interface and cast addresses to that interface.

Recommendation:

For integrations and usability, stick with native datatypes for function inputs and outputs. Here, we can use `address` in the signature and cast internally as needed.

SummerFi:
Commit:
https://github.com/OasisDEX/summer-earn-protocol/pull/228/commits/028da4f15022d6e5fe41bcb1df458b1a3e8e7739

11.8 Undocumented Loss of Voting Power at Maximum Delegation Depth

Context: VotingDecayLibrary.sol#L290-L292

Description:

When a delegation chain reaches `MAX_DELEGATION_DEPTH` (currently set to 2), the implementation returns the original account's decay factor rather than WAD or 0. This creates an edge case where the EIP-5805 invariant "For all accounts a != 0, getVotes(a) SHOULD be the sum of the decayed balances of all accounts that delegate to a" can be violated."

This occurs because at max depth, the decay factor of the original delegator is used, which may be 0 or some other value that doesn't match the expected delegation chain calculation. This means users could unexpectedly lose voting power if they delegate to an account that itself delegates to another account (creating a chain of length > 2).

While having a maximum delegation depth is necessary to prevent recursion attacks, the current behavior at max depth should be explicitly documented to warn users that they may lose voting power if their delegate further delegates to another account.

Recommendation:

Add explicit documentation in the VotingDecayLibrary comments warning users that:

1. There is a maximum delegation depth of 2
2. Delegating to an account that itself delegates will use the original delegator's decay factor at max depth

3. This may result in unexpected voting power calculations if delegation chains exceed the maximum depth

Consider also adding a public view function to help users check delegation chain lengths:

function getDelegationChainLength(address account) external view returns (uint256)

SummerFi:
- Accepted. We've added getDelegationChainLength but will leave voting decay going to zero when depth of 2 is exceeded. This is part of how we designed the incentives to promote participation

11.9 SummerToken: rewardsManager and vestingWalletFactory can be immutable

Context:
SummerToken.sol#L48
SummerToken.sol#L50

Description:
The `rewardsManager` and `vestingWalletFactory` variables are created and assigned in the constructor and have no provision for modification after initialization. These variables can be marked as `immutable` to optimize gas usage and improve code clarity.

Recommendation:
Declare `rewardsManager` and `vestingWalletFactory` as `immutable` to save gas and emphasize their immutability after initialization.

SummerFi:
Commit:
https://github.com/OasisDEX/summer-earn-protocol/pull/228/commits/33d4d79a28def55d67e978005a6b46bd0884a275

11.10 ProtocolAccessManaged: _accessManager can be immutable

Context: ProtocolAccessManaged.sol#L49

Description:
The `_accessManager` variable is assigned in the constructor and does not have a setter, indicating it remains unchanged after initialization. This variable can be marked as `immutable` to optimize gas usage and enhance code clarity.

Recommendation:
Declare `_accessManager` as `immutable` to reduce gas costs and clearly indicate its immutability after initialization.

SummerFi:
Addressed in a prior audit

11.11 ISummerToken does not include all the interfaces from SummerToken

Context: ISummerToken.sol

Description:

The ISummerToken interface does not fully reflect all the externally accessible functions implemented in SummerToken.sol. This divergence between interface and implementation could lead to integration issues and makes the codebase harder to maintain. Missing functions include core ERC20 burning capabilities, cross-chain functionality from OFT, and several important getters.

The issue can be verified by comparing ISummerToken.sol with SummerToken.sol, where several public/external functions are implemented but not declared in the interface.

Recommendation:

Update ISummerToken to include all externally accessible functions from the implementation:

Unset

```
interface ISummerToken is IERC20, IERC20Permit,
ISummerTokenErrors, IVotes {
    // Existing functions remain unchanged
    // Add missing ERC20Burnable functions
    function burn(uint256 amount) external;
    function burnFrom(address account, uint256 amount)
external;

    // Add missing OFT functions
    function send(SendParam memory sendParam, MessagingFee
memory fee, address payable refundAddress) external
payable;
    function estimateSendFee(uint32 dstEid, bytes32 to,
uint256 amount, bool useZro, bytes memory adapterParams)
external view returns (uint256 nativeFee, uint256 zroFee);
    function circulatingSupply() external view returns
(uint256);
    function token() external view returns (address);

    // Add missing getters
    function transferEnableDate() external view returns
(uint256);
    function transfersEnabled() external view returns
(bool);
    function whitelistedAddresses(address account) external
view returns (bool);
    function vestingWalletFactory() external view returns
(address);
    function decayRatePerSecond() external view returns
(uint256);
}
```

This would:

1. Improve interface completeness and accuracy
2. Make the contract's full functionality clear to integrators

SummerFi: Acknowledged

11.12 GovernanceRewardsManager: unstakeOnBehalfOf doc nit

Context: GovernanceRewardsManager.sol#L98

Description:
The function at this location is incorrectly documented as being defined in the `IGovernanceRewardsManager` interface. However, it is actually defined in the `IStakingRewardsManagerBase` interface. This discrepancy in the documentation could lead to confusion for developers and auditors when analyzing the contract and its interface dependencies.

Recommendation:

- Update the documentation to accurately reflect that the function is defined in `IStakingRewardsManagerBase`, not `IGovernanceRewardsManager`.
- Review all interface-related documentation for consistency and correctness to prevent similar errors.
- Add tests to verify the function's behavior aligns with the `IStakingRewardsManagerBase` interface requirements.

SummerFi:

- Commit: https://github.com/OasisDEX/summer-earn-protocol/pull/234/commits/b858c0cd19a5b81015f39607afbca5da658e94d0
- Bricked as we can't account properly for onBehalfOf staking

11.13 General Compiler warnings

Context: Various warnings across multiple files during compilation.

Description:
Compilation warnings indicate unused variables, unused function parameters, and

incorrect function state mutability. These issues may affect code readability, maintainability, and efficiency. Examples include:

1. Unused Local Variables (Warning 2072):
    - `IERC20 underlying` in `AdmiralsQuarters.sol` at line 274.
    - `uint256 underlyingAmount` in `AdmiralsQuarters.sol` at line 279.
    - Similar issues in test files, e.g., `assetsToReceive` in `AdmiralsQuarters.import.t.sol` at line 225.
2. Unused Function Parameters (Warning 5667):
    - `address configurationManager` in `FleetCommanderMock.sol` at line 23.
    - `address configurationManager` in `Tipper.t.sol` at line 215.
3. Function State Mutability (Warning 2018):
    - `_validateToken` in `AdmiralsQuarters.sol` at line 330.
    - `_validateAmount` in `AdmiralsQuarters.sol` at line 334.
    - `_validateRewardsManager` in `AdmiralsQuarters.sol` at line 338.

Recommendation:

1. Unused Local Variables:
    - Remove or utilize the variables to ensure cleaner code.
    - If placeholders for future implementation, add comments indicating their intended purpose.
2. Unused Function Parameters:
    - Remove or comment out unused parameters to suppress warnings.
    - Document their purpose if required for interface compliance or future use.
3. Function State Mutability:
    - Update `view` functions to `pure` if they do not access state variables to improve gas efficiency.
    - Example:

```
Unset
function _validateToken(IERC20 token) internal pure {
    // function logic
}
```

SummerFi:

- Addressed in early audit

11.14 WrappedSummerToken: Hardcoded values in name and symbol

Context: WrappedStakingToken.sol#L13-L20

Description: WrappedStakingToken uses hardcoded name and symbol values to create the name and symbol, respectively. The use of string.concat is therefore unnecessary here as the string can be hardcoded as well.

If the intention was to concat the wrapped + underlyingName and w + underlyingSymbol, the IERC20 interface does not include these values because they are not part of the base ERC20 spec. You can do the following however, which would match the intent:

```
Unset
* @title WrappedStakingToken
 * @notice A simple wrapper for the staking token that
inherits from ERC20Wrapper
 * @dev This contract is used by GovernanceRewardsManager
to wrap staking tokens when they are used as rewards
 */
contract WrappedStakingToken is ERC20Wrapper {
    constructor(
        address underlyingToken
    )
        ERC20(string.concat("Wrapped ",
ERC20(underlyingToken).name()), string.concat("w",
ERC20(underlyingToken).symbol()))
        ERC20Wrapper(IERC20(underlyingToken))
    {}
}
```

Note that this modification also converts the parameter to an address to align with native datatype recommendations.

Recommendation: Remove the unnecessary use of string.concat or use the name/symbol of the underlying token as described above.

SummerFi: Acknowledged

12. Gas Optimization

12.1 SummerVestingWallet: timeBasedVestingAmount can be immutable

Context: SummerVestingWallet.sol#L49

Description:
The `timeBasedVestingAmount` variable is set in the constructor and remains unmodified elsewhere in the contract. This variable can be marked as `immutable` to optimize gas usage and clarify that its value does not change after initialization.

Recommendation:
Declare `timeBasedVestingAmount` as `immutable` to save gas and improve code clarity.

SummerFi: Acknowledged

12.2 SummerVestingWallet: Gas savings on loops

Context:
SummerVestingWallet.sol#L214
SummerVestingWallet.sol#L240

Description:
Gas savings can be achieved by caching the array length before entering a loop, thereby avoiding the cost of recalculating the array size during each iteration. This optimization improves performance, especially for larger arrays.

Example:

```
Unset
uint256 vested = 0;
uint256 _goalLen = goalAmounts.length;
for (uint256 i = 0; i < _goalLen; i++) {
    if (goalsReached[i]) vested += goalAmounts[i];
}
```

Recommendation:
Cache the array length in a local variable before looping to optimize gas usage and improve performance.

SummerFi: Acknowledged

12.3 SummerToken: Unnecessary import of IGovernanceRewardsManager

Context:
SummerToken.sol#L48
SummerToken.sol#L25

Description:
The `IGovernanceRewardsManager` interface is imported but is not strictly necessary in this context. The `GovernanceRewardsManager` code is fully imported and deployed within the constructor, making the interface redundant. Substituting `GovernanceRewardsManager(rewardsManager)` directly in the code for the compiler interface would eliminate this unnecessary import and simplify the code.

Recommendation:

- Remove the `IGovernanceRewardsManager` import from the contract.
- Use `GovernanceRewardsManager(rewardsManager)` directly in the code to access required functionality.
- Verify functionality remains intact after making this change.

SummerFi:

- No longer applicable

12.4 gov-contracts: gas report

Context: Github Gas Report Gist

Description: Gas report for the ProtocolAccessManager.

Recommendation: Rerun the gas report following any updates or amendments.

13. Appendix

13.1 GovernanceRewardsManager: Functions

Readable Functions (Pure/View)

- DECAY_SMOOTHING_FACTOR(): `uint256`
- DECAY_SMOOTHING_FACTOR_BASE(): `uint256`
- balanceOf(address account): `uint256`
- earned(address account, address rewardToken): `uint256`
- generateRole(uint8 roleName, address roleTargetContract): `bytes32`
- getRewardForDuration(address rewardToken): `uint256`
- hasAdmiralsQuartersRole(address account): `bool`
- lastTimeRewardApplicable(address rewardToken): `uint256`
- rewardData(address rewardToken): `uint256, uint256, uint256, uint256, uint256`
- rewardPerToken(address rewardToken): `uint256`
- rewardTokens(uint256 index): `address`
- rewards(address rewardToken, address account): `uint256`
- stakingToken(): `address`
- totalSupply(): `uint256`
- userRewardPerTokenPaid(address rewardToken, address account): `uint256`
- userSmoothedDecayFactor(address account): `uint256`

Writable Functions (Non-Pure/Non-View)

- exit(): ``
- getReward(): ``
- notifyRewardAmount(address rewardToken, uint256 reward, uint256 newRewardsDuration): ``
- removeRewardToken(address rewardToken): ``
- setRewardsDuration(address rewardToken, uint256 _rewardsDuration): ``
- stake(uint256 amount): ``
- stakeOnBehalfOf(address receiver, uint256 amount): ``
- unstake(uint256 amount): ``
- unstakeOnBehalfOf(address , address , uint256): ``
- updateSmoothedDecayFactor(address account): ``

13.2 SummerVestingWallet: Functions

Readable Functions (Pure/View)

- DEFAULT_ADMIN_ROLE(): `bytes32`
- GUARDIAN_ROLE(): `bytes32`
- duration(): `uint256`
- end(): `uint256`
- getRoleAdmin(bytes32 role): `bytes32`
- getVestingType(): `uint8`
- goalAmounts(uint256): `uint256`
- goalsReached(uint256): `bool`
- hasRole(bytes32 role, address account): `bool`
- owner(): `address`
- releasable(address token): `uint256`
- releasable(): `uint256`
- released(): `uint256`
- released(address token): `uint256`
- start(): `uint256`
- supportsInterface(bytes4 interfaceId): `bool`
- timeBasedVestingAmount(): `uint256`
- token(): `address`
- vestedAmount(uint64 timestamp): `uint256`
- vestedAmount(address token, uint64 timestamp): `uint256`

Writable Functions (Non-Pure/Non-View)

- addNewGoal(uint256 goalAmount): ``
- grantRole(bytes32 role, address account): ``
- markGoalReached(uint256 goalNumber): ``
- recallUnvestedTokens(): ``
- release(address token): ``
- release(): ``
- renounceOwnership(): ``
- renounceRole(bytes32 role, address callerConfirmation): ``
- revokeRole(bytes32 role, address account): ``
- transferOwnership(address newOwner): ``

13.3 SummerGovernor: Functions

Readable Functions (Pure/View)

- BALLOT_TYPEHASH(): `bytes32`
- CLOCK_MODE(): `string`
- COUNTING_MODE(): `string`
- EXTENDED_BALLOT_TYPEHASH(): `bytes32`
- MAX_PROPOSAL_THRESHOLD(): `uint256`
- MIN_PROPOSAL_THRESHOLD(): `uint256`
- allowInitializePath(tuple origin): `bool`
- clock(): `uint48`
- config(): `address`
- eip712Domain(): `bytes1, string, string, uint256, address, bytes32, uint256[]`
- endpoint(): `address`
- getVotes(address account, uint256 timepoint): `uint256`
- getVotesWithParams(address account, uint256 timepoint, bytes params): `uint256`
- getWhitelistAccountExpiration(address account): `uint256`
- getWhitelistGuardian(): `address`
- hasVoted(uint256 proposalId, address account): `bool`
- hashProposal(address[] targets, uint256[] values, bytes[] calldatas, bytes32 descriptionHash): `uint256`

- isComposeMsgSender(tuple , bytes , address _sender): `bool`
- isWhitelisted(address account): `bool`
- name(): `string`
- nextNonce(uint32 , bytes32): `uint64`
- nonces(address owner): `uint256`
- oAppVersion(): `uint64, uint64`
- owner(): `address`
- peers(uint32 eid): `bytes32`
- proposalChainId(): `uint32`
- proposalDeadline(uint256 proposalId): `uint256`
- proposalEta(uint256 proposalId): `uint256`
- proposalNeedsQueuing(uint256 proposalId): `bool`
- proposalProposer(uint256 proposalId): `address`
- proposalSnapshot(uint256 proposalId): `uint256`
- proposalThreshold(): `uint256`
- proposalVotes(uint256 proposalId): `uint256, uint256, uint256`
- quorum(uint256 timepoint): `uint256`
- quorumDenominator(): `uint256`
- quorumNumerator(uint256 timepoint): `uint256`
- quorumNumerator(): `uint256`
- state(uint256 proposalId): `uint8`
- supportsInterface(bytes4 interfaceId): `bool`
- timelock(): `address`
- token(): `address`
- version(): `string`
- votingDelay(): `uint256`
- votingPeriod(): `uint256`

Writable Functions (Non-Pure/Non-View)

- cancel(address[] targets, uint256[] values, bytes[] calldatas, bytes32 descriptionHash): `uint256`
- castVote(uint256 proposalId, uint8 support): `uint256`
- castVoteBySig(uint256 proposalId, uint8 support, address voter, bytes signature): `uint256`
- castVoteWithReason(uint256 proposalId, uint8 support, string reason): `uint256`

- castVoteWithReasonAndParams(uint256 proposalId, uint8 support, string reason, bytes params): `uint256`
- castVoteWithReasonAndParamsBySig(uint256 proposalId, uint8 support, address voter, string reason, bytes params, bytes signature): `uint256`
- execute(address[] targets, uint256[] values, bytes[] calldatas, bytes32 descriptionHash): `uint256`
- lzReceive(tuple _origin, bytes32 _guid, bytes _message, address _executor, bytes _extraData): ``
- onERC1155BatchReceived(address , address , uint256[] , uint256[] , bytes): `bytes4`
- onERC1155Received(address , address , uint256 , uint256 , bytes): `bytes4`
- onERC721Received(address , address , uint256 , bytes): `bytes4`
- propose(address[] targets, uint256[] values, bytes[] calldatas, string description): `uint256`
- queue(address[] targets, uint256[] values, bytes[] calldatas, bytes32 descriptionHash): `uint256`
- relay(address target, uint256 value, bytes data): ``
- renounceOwnership(): ``
- sendProposalToTargetChain(uint32 _dstEid, address[] _dstTargets, uint256[] _dstValues, bytes[] _dstCalldatas, bytes32 _dstDescriptionHash, bytes _options): ``
- setDelegate(address _delegate): ``
- setPeer(uint32 _eid, bytes32 _peer): ``
- setProposalThreshold(uint256 newProposalThreshold): ``
- setVotingDelay(uint48 newVotingDelay): ``
- setVotingPeriod(uint32 newVotingPeriod): ``
- setWhitelistAccountExpiration(address account, uint256 expiration): ``
- setWhitelistGuardian(address _whitelistGuardian): ``
- transferOwnership(address newOwner): ``
- updateQuorumNumerator(uint256 newQuorumNumerator): ``
- updateTimelock(address newTimelock): ``

13.4 SummerToken: Functions

Readable Functions (Pure/View)

- CLOCK_MODE(): `string`
- DOMAIN_SEPARATOR(): `bytes32`
- SEND(): `uint16`

- SEND_AND_CALL(): `uint16`
- allowInitializePath(tuple origin): `bool`
- allowance(address owner, address spender): `uint256`
- approvalRequired(): `bool`
- balanceOf(address account): `uint256`
- cap(): `uint256`
- checkpoints(address account, uint32 pos): `tuple`
- clock(): `uint48`
- combineOptions(uint32 _eid, uint16 _msgType, bytes _extraOptions): `bytes`
- decimalConversionRate(): `uint256`
- decimals(): `uint8`
- delegates(address account): `address`
- eip712Domain(): `bytes1, string, string, uint256, address, bytes32, uint256[]`
- endpoint(): `address`
- enforcedOptions(uint32 eid, uint16 msgType): `bytes`
- generateRole(uint8 roleName, address roleTargetContract): `bytes32`
- getDecayFactor(address account): `uint256`
- getDecayFreeWindow(): `uint40`
- getPastTotalSupply(uint256 timepoint): `uint256`
- getPastVotes(address account, uint256 timepoint): `uint256`
- getVotes(address account): `uint256`
- hasAdmiralsQuartersRole(address account): `bool`
- isComposeMsgSender(tuple , bytes , address _sender): `bool`
- isPeer(uint32 _eid, bytes32 _peer): `bool`
- msgInspector(): `address`
- name(): `string`
- nextNonce(uint32 , bytes32): `uint64`
- nonces(address owner): `uint256`
- numCheckpoints(address account): `uint32`
- oApp(): `address`
- oAppVersion(): `uint64, uint64`
- oftVersion(): `bytes4, uint64`
- owner(): `address`
- peers(uint32 eid): `bytes32`

- preCrime(): `address`
- quoteOFT(tuple _sendParam): `tuple, tuple[], tuple`
- quoteSend(tuple _sendParam, bool _payInLzToken): `tuple`
- rewardsManager(): `address`
- sharedDecimals(): `uint8`
- symbol(): `string`
- token(): `address`
- totalSupply(): `uint256`
- transferEnableDate(): `uint256`
- transfersEnabled(): `bool`
- vestingWalletFactory(): `address`
- whitelistedAddresses(address account): `bool`

Writable Functions (Non-Pure/Non-View)

- addToWhitelist(address account): ``
- approve(address spender, uint256 value): `bool`
- burn(uint256 value): ``
- burnFrom(address account, uint256 value): ``
- delegate(address delegatee): ``
- delegateBySig(address delegatee, uint256 nonce, uint256 expiry, uint8 v, bytes32 r, bytes32 s): ``
- enableTransfers(): ``
- lzReceive(tuple _origin, bytes32 _guid, bytes _message, address _executor, bytes _extraData): ``
- lzReceiveAndRevert(tuple[] _packets): ``
- lzReceiveSimulate(tuple _origin, bytes32 _guid, bytes _message, address _executor, bytes _extraData): ``
- permit(address owner, address spender, uint256 value, uint256 deadline, uint8 v, bytes32 r, bytes32 s): ``
- removeFromWhitelist(address account): ``
- renounceOwnership(): ``
- send(tuple _sendParam, tuple _fee, address _refundAddress): `tuple, tuple`
- setDecayFreeWindow(uint40 newWindow): ``
- setDecayFunction(uint8 newFunction): ``
- setDecayRatePerSecond(uint256 newRatePerSecond): ``
- setDelegate(address _delegate): ``
- setEnforcedOptions(tuple[] _enforcedOptions): ``

- setMsgInspector(address _msgInspector): ``
- setPeer(uint32 _eid, bytes32 _peer): ``
- setPreCrime(address _preCrime): ``
- transfer(address to, uint256 value): `bool`
- transferFrom(address from, address to, uint256 value): `bool`
- transferOwnership(address newOwner): ``
- updateDecayFactor(address account): ``
