



Security Assessment

**GarfieldFinance**

May 29th, 2021



# Table of Contents

## Summary

### Overview

- Project Summary
- Audit Summary
- Vulnerability Summary
- Audit Scope

### Findings

- GLOBAL-01 : External Dependency
- CCK-01 : Lack of Input Validation
- CCK-02 : Lack of Sanity Check
- CCK-03 : Centralized Risks
- CCK-04 : Boolean Equality
- CCK-05 : Unlocked Compiler Version Declaration
- CCK-06 : Unknown Converter
- CCK-07 : Missing Emit Events
- CCK-08 : Proper Usage of “public” and “external” type
- CCK-09 : Confusing Log Message in Require Checking
- SCC-01 : Lack of Input Validation
- SCC-02 : Missing Emit Events
- SCC-03 : Proper Usage of “public” and “external” type
- SCC-04 : No Restriction on Setting of Sensitive Variable
- SCC-05 : Unhandled Return Value
- SLH-01 : Lack of Input Validation
- SLH-02 : Missing Emit Events
- SLH-03 : Proper Usage of “public” and “external” type
- SLH-04 : No Restriction on Setting of Sensitive Variable
- SLH-05 : Unhandled Return Value
- VCK-01 : Lack of Input Validation
- VCK-02 : Unlocked Compiler Version Declaration
- VCK-03 : Missing Emit Events
- VCK-04 : Proper Usage of “public” and “external” type
- VCK-05 : No Restriction on Setting of Sensitive Variable
- VCK-06 : Constrain Times of Function Calls
- VHT-01 : Lack of Input Validation

VHT-02 : Unlocked Compiler Version Declaration

VHT-03 : Missing Emit Events

VHT-04 : Proper Usage of “public” and “external” type

VHT-05 : No Restriction on Setting of Sensitive Variable

VHT-06 : Constrain Times of Function Calls

## **Appendix**

### **Disclaimer**

### **About**

# Summary

This report has been prepared for GarfieldFinance smart contracts, to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Manual Review and Static Analysis techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases given they are currently missing in the repository;
- Provide more comments per each function for readability, especially contracts are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

# Overview

## Project Summary

Project Name	GarfieldFinance
Platform	Ethereum
Language	Solidity
Codebase	<a href="https://github.com/GarfieldLab/garfieldfinance-Valut">https://github.com/GarfieldLab/garfieldfinance-Valut</a>
Commits	<70dd1cd28304447e062004067e0ad7c6eee3a76e>

## Audit Summary

Delivery Date	May 29, 2021
Audit Methodology	Manual Review, Static Analysis
Key Components	

## Vulnerability Summary

Total Issues	32
● Critical	1
● Major	0
● Medium	0
● Minor	13
● Informational	18
● Discussion	0

## Audit Scope

ID	file	SHA256 Checksum
CCK	Controller.sol	59dc204b46cf0b901d68f0d38b7d37b2337539e9d47000489427b51bab4b301f
SCC	StrategyChannels.sol	be2e69b408a2d600c03c8db157156deb946685716991936fdf737caa16fc29bd
SLH	StrategyLendHub.sol	75026c54e6a5471eb92d76bfb94c159b4d8ca89d0c2a7fb85e9664c6ee5e17f9
VCK	gVault.sol	452a7e7aef469ed5dc65fa1cd8bcdcb75194588cc00a1c843721505cee83af72
VHT	gVaultHT.sol	376856d12252a84c94ff61c76cd169cd9ad59629912ac44f9ce87f294c402476

## System Overview

Garfield Finance is a decentralized investment aggregator. The core components are the Controller, Strategy and Vault. Vaults are simple token containers and function, tracking user's share of an ever growing pool. Controller is the control mechanism managed by governance. Strategies are simplistic containers, they specify what they “want”, and their job is to maximize returns on that asset however they can, in a lossless way. All user deposits are immediately transferred to a third-party service (like compound), in which interests are earned. Also when withdraw happens, the strategy will use MDEX to swap between tokens, and earn MDEX rewards. The system should only be used if the service is appropriately trusted. And these external protocols are not in the scope of this audit.

## Centralized Risk

To bridge the trust gap between the administrator and users, the administrator needs to express a sincere attitude with the consideration of the administrator team’s anonymousness. The administrator has the responsibility to notify users with the following capability of the administrator:

- Administrators can transfer amount to admin via `Controller.inCaseTokensGetStuck(address _token, uint256 _amount)` function.

The advantage of `inCaseTokensGetStuck()` method in the protocol is that the administrator reserves the ability to rescue the assets in this contract under unexpected cases. It is also worthy of note the downside of `inCaseTokensGetStuck` method, where the treasury in this contract can be migrated to admin.

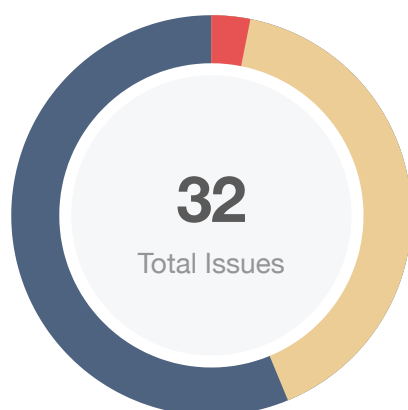
- Administrators can change the strategy for certain token forcibly via `Controller.setStrategyWithoutWithdraw()` function, without withdraw the investments in the strategy.

To improve the trustworthiness of the project, any dynamic runtime updates in the project should be notified to the community. Any plan to invoke the above-mentioned functions should be also considered to move to the execution queue of the Timelock contract.

## Financial Models

Financial models of blockchain protocols need to be resilient to attacks. It needs to pass simulations and verifications to guarantee the security of the overall protocol.

# Findings



Critical	1 (3.13%)
Major	0 (0.00%)
Medium	0 (0.00%)
Minor	13 (40.63%)
Informational	18 (56.25%)
Discussion	0 (0.00%)

ID	Title	Category	Severity	Status
GLOBAL-01	External Dependency	Data Flow	Minor	ⓘ Acknowledged
CCK-01	Lack of Input Validation	Volatile Code	Minor	ⓘ Acknowledged
CCK-02	Lack of Sanity Check	Volatile Code	Minor	ⓘ Acknowledged
CCK-03	Centralized Risks	Centralization / Privilege	Critical	ⓘ Acknowledged
CCK-04	Boolean Equality	Gas Optimization	Informational	ⓘ Acknowledged
CCK-05	Unlocked Compiler Version Declaration	Language Specific	Informational	ⓘ Acknowledged
CCK-06	Unknown Converter	Data Flow	Informational	ⓘ Acknowledged
CCK-07	Missing Emit Events	Gas Optimization	Informational	ⓘ Acknowledged
CCK-08	Proper Usage of “public” and “external” type	Coding Style	Informational	ⓘ Acknowledged
CCK-09	Confusing Log Message in Require Checking	Coding Style	Informational	ⓘ Acknowledged



ID	Title	Category	Severity	Status
SCC-01	Lack of Input Validation	Volatile Code	Minor	① Acknowledged
SCC-02	Missing Emit Events	Gas Optimization	Informational	① Acknowledged
SCC-03	Proper Usage of “public” and “external” type	Coding Style	Informational	① Acknowledged
SCC-04	No Restriction on Setting of Sensitive Variable	Control Flow	Minor	① Acknowledged
SCC-05	Unhandled Return Value	Logical Issue	Informational	① Acknowledged
SLH-01	Lack of Input Validation	Volatile Code	Minor	① Acknowledged
SLH-02	Missing Emit Events	Gas Optimization	Informational	① Acknowledged
SLH-03	Proper Usage of “public” and “external” type	Coding Style	Informational	① Acknowledged
SLH-04	No Restriction on Setting of Sensitive Variable	Control Flow	Minor	① Acknowledged
SLH-05	Unhandled Return Value	Logical Issue	Informational	① Acknowledged
VCK-01	Lack of Input Validation	Volatile Code	Minor	① Acknowledged
VCK-02	Unlocked Compiler Version Declaration	Language Specific	Informational	① Acknowledged
VCK-03	Missing Emit Events	Gas Optimization	Informational	① Acknowledged
VCK-04	Proper Usage of “public” and “external” type	Coding Style	Informational	① Acknowledged
VCK-05	No Restriction on Setting of Sensitive Variable	Control Flow	Minor	① Acknowledged
VCK-06	Constrain Times of Function Calls	Gas Optimization	Minor	① Acknowledged

ID	Title	Category	Severity	Status
VHT-01	Lack of Input Validation	Volatile Code	● Minor	① Acknowledged
VHT-02	Unlocked Compiler Version Declaration	Language Specific	● Informational	① Acknowledged
VHT-03	Missing Emit Events	Gas Optimization	● Informational	① Acknowledged
VHT-04	Proper Usage of “public” and “external” type	Coding Style	● Informational	① Acknowledged
VHT-05	No Restriction on Setting of Sensitive Variable	Control Flow	● Minor	① Acknowledged
VHT-06	Constrain Times of Function Calls	Gas Optimization	● Minor	① Acknowledged

## GLOBAL-01 | External Dependency

Category	Severity	Location	Status
Data Flow	● Minor	Global	ⓘ Acknowledged

### Description

This protocol has external dependencies. The functions in `cToken`, `CETH`, `IUnitroller`, `Uni`, `ISwapMining` interfaces are all from third-party contracts, which are not in the scope of this audit.

### Recommendation

Make sure the third-party implementations and the way these functions are called can meet the requirements.

## CCK-01 | Lack of Input Validation

Category	Severity	Location	Status
Volatile Code	● Minor	Controller.sol: 551~552(Controller)	ⓘ Acknowledged

### Description

Missing validation for the input variables `_router`, `_rewards` in function `Controller.constructor()`; `_token`, `_controller` in function `gVault.constructor()`; `_controller`, `_ctoken`, `_want`, `_comprtl`, `_comp` in function `StrategyCommon.constructor()`.

### Recommendation

Consider adding below checks to ensure these input variables are not equal to `address(0)`:

```
require(_router != address(0), "Controller: _router is zero address");
router = _router;
require(_rewards != address(0), "Controller: _rewards is zero address");
rewards = _rewards;
```

## CCK-02 | Lack of Sanity Check

Category	Severity	Location	Status
Volatile Code	● Minor	Controller.sol: 695(Controller), 714(Controller), 837(Controller)	ⓘ Acknowledged

### Description

There's no checking for the existence of strategy for given token.

### Recommendation

Consider to check the existence of the strategy for any given token with following code snippets:

```
require(strategies[_token] != address(0), "strategy does not exist");
```

## CCK-03 | Centralized Risks

Category	Severity	Location	Status
Centralization / Privilege	● Critical	Controller.sol: 661(Controller), 743(Controller)	ⓘ Acknowledged

### Description

To bridge the trust gap between the administrator and users, the administrator needs to express a sincere attitude with the consideration of the administrator team's anonymousness. The administrator has the responsibility to notify users with the following capability of the administrator:

- Administrators can transfer amount to admin via `Controller.inCaseTokensGetStuck()` function.

The advantage of `inCaseTokensGetStuck()` method in the protocol is that the administrator reserves the ability to rescue the assets in this contract under unexpected cases. It is also worthy of note the downside of `inCaseTokensGetStuck` method, where the treasury in this contract can be migrated to admin.

- Administrators can change the strategy for certain token forcibly via `Controller.setStrategyWithoutWithdraw()` function, without withdraw the investments in the strategy.

### Recommendation

To improve the trustworthiness of the project, any dynamic runtime updates in the project should be notified to the community. Any plan to invoke the above-mentioned functions should be also considered to move to the execution queue of the Timelock contract.

### Alleviation

The development team heeded our advice and removed `Controller.setStrategyWithoutWithdraw()` function in commit `0c56471b57fbf2437707dfaf17229bbee4de690d`. The development team stated: "we will move the ownership of this contract to Timelock to restrict the usage of function `Controller.inCaseTokenGetStuck()`".

## CCK-04 | Boolean Equality

Category	Severity	Location	Status
Gas Optimization	● Informational	Controller.sol: 663(Controller), 675(Controller)	ⓘ Acknowledged

### Description

Boolean constants can be used directly and do not need to be compared to true or false.

Example:

```
require(approvedStrategies[_token][_strategy] == true, "!approved");
```

### Recommendation

Consider changing it as following example:

```
require(approvedStrategies[_token][_strategy], "!approved");
```

## CCK-05 | Unlocked Compiler Version Declaration

Category	Severity	Location	Status
Language Specific	● Informational	Controller.sol: 5, 84, 245, 388, 463, 479, 487, 513	ⓘ Acknowledged

### Description

The compiler version utilized throughout the project uses ' $\geq 0.6.12$ ', ' $\wedge 0.6.0$ ', ' $\wedge 0.6.12$ ', ' $\wedge 0.6.2$ ', denoting that a compiler version which is greater than the version will be used to compile the contracts.

### Recommendation

It is a general practice to instead lock the compiler at a specific version rather than allow a range of compiler versions to be utilized to avoid compiler-specific bugs and be able to identify ones more easily. We recommend locking the compiler at the lowest possible version that supports all the capabilities wished by the codebase. This will ensure that the project utilizes a compiler version that has been in use for the longest time and as such is less likely to contain yet-undiscovered bugs.



## CCK-06 | Unknown Converter

Category	Severity	Location	Status
Data Flow	● Informational	Controller.sol: 700(Controller)	① Acknowledged

### Description

Function `earn()` is calling external implementations.

```
_amount = IConverter(converter).convert(_strategy);
```

The address `converter` is set by setter function, so the actual implementation is out of the audit scope.

### Recommendation

Make sure that it supports all the wanted tokens for all strategies.

## CCK-07 | Missing Emit Events

Category	Severity	Location	Status
Gas Optimization	● Informational	Controller.sol: 570(Controller), 580(Controller), 590(Controller), 600(Controller)	ⓘ Acknowledged

### Description

Several sensitive actions are defined without event declarations.

1. Functions `setController()`, `setStrategist()`, `setRouter()`, `setKeeper()`, `setGovernance()` can change the governance of the contracts.
2. Functions `setMin()`, `setSplit()`, `setWithdrawalFee()`, `setHarvestReward()`, `setStrategistReward()` will decide amounts that are critical to the project.

### Recommendation

Consider adding events for sensitive actions, and emit it in the function.

## CCK-08 | Proper Usage of “public” and “external” type

Category	Severity	Location	Status
Coding Style	● Informational	Controller.sol: 560, 570, 580, 590, 600, 611, 623, 634, 646, 661, 673, 723, 733, 755, 766, 784, 836	ⓘ Acknowledged

### Description

`public` functions that are never called by the contract could be declared `external`. When the inputs are arrays, `external` functions are more efficient than `public` functions.

For example: `setRewards()`, `setStrategist()`, `setSplit()`, `setRouter()`, `setGovernance()`, `setVault()`, `approveStrategy()`, `revokeStrategy()`, `setConverter()`, `setStrategyWithoutWithdraw()`, `setStrategy()`, `withdrawAll()`, `withdrawAllFromStrategy()`, `inCaseStrategyTokenGetStuck()`, `getExpectedReturn()`, `yearn()`, `withdraw()` in contract `Controller`. `setGovernance()`, `setController()` in contract `VaultCommon`. `getPricePerFullShare()` in contract `gVault`. `getPricePerFullShare()` in contract `gVaultHT`. `balanceOf()`, `withdraw()`, `harvest()` in contract `StrategyCommon`.

### Recommendation

Consider using the `external` attribute for functions never called from the contract.

## CCK-09 | Confusing Log Message in Require Checking

Category	Severity	Location	Status
Coding Style	● Informational	Controller.sol: 613(Controller)	ⓘ Acknowledged

### Description

The log message in require statements is hard to be understood by reader. For example

```
require(vaults[_token] == address(0), "vault");
```

### Recommendation

Consider to improve readability of log messages in require statements. For example:

```
require(vaults[_token] == address(0), "The vault of token has been set");
```

## SCC-01 | Lack of Input Validation

Category	Severity	Location	Status
Volatile Code	● Minor	StrategyChannels.sol: 432~436(StrategyCommon)	📄 Acknowledged

### Description

Missing validation for the input variables `_router`, `_rewards` in function `Controller.constructor()`; `_token`, `_controller` in function `gVault.constructor()`; `_controller`, `_ctoken`, `_want`, `_comptrol`, `_comp` in function `StrategyCommon.constructor()`.

### Recommendation

Consider adding below checks to ensure these input variables are not equal to `address(0)`:

```
require(_router != address(0), "Controller: _router is zero address");
router = _router;
require(_rewards != address(0), "Controller: _rewards is zero address");
rewards = _rewards;
```

## SCC-02 | Missing Emit Events

Category	Severity	Location	Status
Gas Optimization	● Informational	StrategyChannels.sol: 450(StrategicCommon), 450(StrategicCommon), 460(StrategicCommon), 470(StrategicCommon), 480(StrategicCommon), 490(StrategicCommon), 496(StrategicCommon), 502 (StrategicCommon)	ⓘ Acknowledged

### Description

Several sensitive actions are defined without event declarations.

1. Functions `setController()`, `setStrategist()`, `setRouter()`, `setKeeper()`, `setGovernance()` can change the governance of the contracts.
2. Functions `setMin()`, `setSplit()`, `setWithdrawalFee()`, `setHarvestReward()`, `setStrategistReward()` will decide amounts that are critical to the project.

### Recommendation

Consider adding events for sensitive actions, and emit it in the function.

## SCC-03 | Proper Usage of “public” and “external” type

Category	Severity	Location	Status
Coding Style	● Informational	StrategyChannels.sol: 521, 622, 742	📄 Acknowledged

### Description


`public` functions that are never called by the contract could be declared `external`. When the inputs are arrays, `external` functions are more efficient than `public` functions.

For example: `setRewards()`, `setStrategist()`, `setSplit()`, `setRouter()`, `setGovernance()`, `setVault()`, `approveStrategy()`, `revokeStrategy()`, `setConverter()`, `setStrategyWithoutWithdraw()`, `setStrategy()`, `withdrawAll()`, `withdrawAllFromStrategy()`, `inCaseStrategyTokenGetStuck()`, `getExpectedReturn()`, `yearn()`, `withdraw()` in contract `Controller`. `setGovernance()`, `setController()` in contract `VaultCommon`. `getPricePerFullShare()` in contract `gVault`. `getPricePerFullShare()` in contract `gVaultHT`. `balanceOf()`, `withdraw()`, `harvest()` in contract `StrategyCommon`.

### Recommendation

Consider using the `external` attribute for functions never called from the contract.

## SCC-04 | No Restriction on Setting of Sensitive Variable

Category	Severity	Location	Status
Control Flow	Minor	StrategyChannels.sol: 460(VaultCommon), 470(VaultCommon), 480(VaultCommon)	 Acknowledged

### Description

The value of `min`, `withdrawalFee`, `strategistReward`, `harvestReward` can be arbitrarily updated by the `governance` role, which initially is the contract deployer. For example, `min` should always be less than or equal to the value of `max`.

### Recommendation

Consider to enforce any changes on the values of these variables to be restricted by Timelock, Multisig, or other DAO mechanism. Meanwhile, add following check on `min` to guarantee its value is always less than or equal to `max`.

```
require(_min <= max, "_min is over max");
```



## SCC-05 | Unhandled Return Value

Category	Severity	Location	Status
Logical Issue	● Informational	StrategyChannels.sol: 551(StrategicCommon), 553(StrategicCommon), 574(StrategicCommon), 575(StrategicCommon), 606(StrategicCommon), 607(StrategicCommon), 634(StrategicCommon), 662(StrategicCommon), 669(StrategicCommon), 721(StrategicCommon)	ⓘ Acknowledged

### Description

The token's `transfer`, `transferFrom`, `approve` is not void-returning functions. Ignoring the return value might cause some unexpected exception, especially if the callee function doesn't revert automatically when failing.

### Recommendation

Consider checking the output of the aforementioned functions before continuing processing.

## SLH-01 | Lack of Input Validation

Category	Severity	Location	Status
Volatile Code	● Minor	StrategyLendHub.sol: 432~436(StrategyCommon)	📄 Acknowledged

### Description

Missing validation for the input variables `_router`, `_rewards` in function `Controller.constructor()`; `_token`, `_controller` in function `gVault.constructor()`; `_controller`, `_ctoken`, `_want`, `_comptrol`, `_comp` in function `StrategyCommon.constructor()`.

### Recommendation

Consider adding below checks to ensure these input variables are not equal to `address(0)`:

```
require(_router != address(0), "Controller: _router is zero address");
router = _router;
require(_rewards != address(0), "Controller: _rewards is zero address");
rewards = _rewards;
```

## SLH-02 | Missing Emit Events

Category	Severity	Location	Status
Gas Optimization	● Informational	StrategyLendHub.sol: 450(StrategyCommon), 460(StrategyCommon), 470(StrategyCommon), 480(StrategyCommon), 490(StrategyCommon), 496(StrategyCommon), 502(StrategyCommon)	ⓘ Acknowledged

### Description

Several sensitive actions are defined without event declarations.

1. Functions `setController()`, `setStrategist()`, `setRouter()`, `setKeeper()`, `setGovernance()` can change the governance of the contracts.
2. Functions `setMin()`, `setSplit()`, `setWithdrawalFee()`, `setHarvestReward()`, `setStrategistReward()` will decide amounts that are critical to the project.

### Recommendation

Consider adding events for sensitive actions, and emit it in the function.

## SLH-03 | Proper Usage of “public” and “external” type

Category	Severity	Location	Status
Coding Style	● Informational	StrategyLendHub.sol: 521, 622, 742	📄 Acknowledged

### Description


`public` functions that are never called by the contract could be declared `external`. When the inputs are arrays, `external` functions are more efficient than `public` functions.

For example: `setRewards()`, `setStrategist()`, `setSplit()`, `setRouter()`, `setGovernance()`, `setVault()`, `approveStrategy()`, `revokeStrategy()`, `setConverter()`, `setStrategyWithoutWithdraw()`, `setStrategy()`, `withdrawAll()`, `withdrawAllFromStrategy()`, `inCaseStrategyTokenGetStuck()`, `getExpectedReturn()`, `yearn()`, `withdraw()` in contract `Controller`. `setGovernance()`, `setController()` in contract `VaultCommon`. `getPricePerFullShare()` in contract `gVault`. `getPricePerFullShare()` in contract `gVaultHT`. `balanceOf()`, `withdraw()`, `harvest()` in contract `StrategyCommon`.

### Recommendation

Consider using the `external` attribute for functions never called from the contract.

## SLH-04 | No Restriction on Setting of Sensitive Variable

Category	Severity	Location	Status
Control Flow	Minor	StrategyLendHub.sol: 460(VaultCommon), 470(VaultCommon), 480(VaultCommon)	 Acknowledged

### Description

The value of `min`, `withdrawalFee`, `strategistReward`, `harvestReward` can be arbitrarily updated by the `governance` role, which initially is the contract deployer. For example, `min` should always be less than or equal to the value of `max`.

### Recommendation

Consider to enforce any changes on the values of these variables to be restricted by Timelock, Multisig, or other DAO mechanism. Meanwhile, add following check on `min` to guarantee its value is always less than or equal to `max`.

```
require(_min <= max, "_min is over max");
```

## SLH-05 | Unhandled Return Value

Category	Severity	Location	Status
Logical Issue	● Informational	StrategyLendHub.sol: 551(StrategicCommon), 553(StrategicCommon), 574(StrategicCommon), 575(StrategicCommon), 606(StrategicCommon), 607(StrategicCommon), 634(StrategicCommon), 662(StrategicCommon), 669(StrategicCommon), 721(StrategicCommon)	ⓘ Acknowledged


### Description

The token's `transfer`, `transferFrom`, `approve` is not void-returning functions. Ignoring the return value might cause some unexpected exception, especially if the callee function doesn't revert automatically when failing.

### Recommendation

Consider checking the output of the aforementioned functions before continuing processing.

## VCK-01 | Lack of Input Validation

Category	Severity	Location	Status
Volatile Code	Minor	gVault.sol: 906~908(gVault)	 Acknowledged

### Description

Missing validation for the input variables `_router`, `_rewards` in function `Controller.constructor()`; `_token`, `_controller` in function `gVault.constructor()`; `_controller`, `_ctoken`, `_want`, `_comptrl`, `_comp` in function `StrategyCommon.constructor()`.

### Recommendation

Consider adding below checks to ensure these input variables are not equal to `address(0)`:

```
require(_router != address(0), "Controller: _router is zero address");
router = _router;
require(_rewards != address(0), "Controller: _rewards is zero address");
rewards = _rewards;
```

## VCK-02 | Unlocked Compiler Version Declaration

Category	Severity	Location	Status
Language Specific	● Informational	gVault.sol: 5, 166, 245, 388, 464, 490, 797, 819, 892	ⓘ Acknowledged

### Description

The compiler version utilized throughout the project uses ' $\geq 0.6.12$ ', ' $\wedge 0.6.0$ ', ' $\wedge 0.6.12$ ', ' $\wedge 0.6.2$ ', denoting that a compiler version which is greater than the version will be used to compile the contracts.

### Recommendation

It is a general practice to instead lock the compiler at a specific version rather than allow a range of compiler versions to be utilized to avoid compiler-specific bugs and be able to identify ones more easily. We recommend locking the compiler at the lowest possible version that supports all the capabilities wished by the codebase. This will ensure that the project utilizes a compiler version that has been in use for the longest time and as such is less likely to contain yet-undiscovered bugs.



## VCK-03 | Missing Emit Events

Category	Severity	Location	Status
Gas Optimization	● Informational	gVault.sol: 849(VaultCommon), 855(VaultCommon), 861(VaultCommon)	ⓘ Acknowledged

### Description

Several sensitive actions are defined without event declarations.

1. Functions `setController()`, `setStrategist()`, `setRouter()`, `setKeeper()`, `setGovernance()` can change the governance of the contracts.
2. Functions `setMin()`, `setSplit()`, `setWithdrawalFee()`, `setHarvestReward()`, `setStrategistReward()` will decide amounts that are critical to the project.

### Recommendation

Consider adding events for sensitive actions, and emit it in the function.

## VCK-04 | Proper Usage of “public” and “external” type

Category	Severity	Location	Status
Coding Style	● Informational	gVault.sol: 855, 861, 997	📄 Acknowledged

### Description

`public` functions that are never called by the contract could be declared `external`. When the inputs are arrays, `external` functions are more efficient than `public` functions.

For example: `setRewards()`, `setStrategist()`, `setSplit()`, `setRouter()`, `setGovernance()`, `setVault()`, `approveStrategy()`, `revokeStrategy()`, `setConverter()`, `setStrategyWithoutWithdraw()`, `setStrategy()`, `withdrawAll()`, `withdrawAllFromStrategy()`, `inCaseStrategyTokenGetStuck()`, `getExpectedReturn()`, `yearn()`, `withdraw()` in contract `Controller`. `setGovernance()`, `setController()` in contract `VaultCommon`. `getPricePerFullShare()` in contract `gVault`. `getPricePerFullShare()` in contract `gVaultHT`. `balanceOf()`, `withdraw()`, `harvest()` in contract `StrategyCommon`.

### Recommendation

Consider using the `external` attribute for functions never called from the contract.

## VCK-05 | No Restriction on Setting of Sensitive Variable

Category	Severity	Location	Status
Control Flow	● Minor	gVault.sol: 851(VaultCommon)	① Acknowledged

### Description

The value of `min`, `withdrawalFee`, `strategistReward`, `harvestReward` can be arbitrarily updated by the `governance` role, which initially is the contract deployer. For example, `min` should always be less than or equal to the value of `max`.

### Recommendation

Consider to enforce any changes on the values of these variables to be restricted by Timelock, Multisig, or other DAO mechanism. Meanwhile, add following check on `min` to guarantee its value is always less than or equal to `max`.

```
require(_min <= max, "_min is over max");
```

## VCK-06 | Constrain Times of Function Calls

Category	Severity	Location	Status
Gas Optimization	● Minor	gVault.sol: 945(gVault)	ⓘ Acknowledged

### Description

In `deposit()` function of `gVault` and `gVaultHT`, no matter how much the deposit amount is, the function will always call `earn()` method transferring token to strategy via controller. This process will consume lots of gas.

### Recommendation

Consider add a variable like `earnLowerlimit` as a threshold of the deposit amount, and corresponding setter function. `earn()` function will be called only if the balance of the vault reaches the value of `earnLowerlimit`.

For example:


```
if (token.balanceOf(address(this)) > earnLowerlimit){  
    earn();  
}
```

Additionally, the variable `min` equals to 10000 currently. Hence the `available()` will be 100% of the balance. In case user want to withdraw, the controller needs to withdraw from the strategy, which will waste gas.

```
uint256 public min = 10000;
```

Consider to set `min` to 9500.

## VHT-01 | Lack of Input Validation

Category	Severity	Location	Status
Volatile Code	Minor	gVaultHT.sol: 926(gVaultHT)	 Acknowledged

### Description

Missing validation for the input variables `_router`, `_rewards` in function `Controller.constructor()`; `_token`, `_controller` in function `gVault.constructor()`; `_controller`, `_ctoken`, `_want`, `_comprtl`, `_comp` in function `StrategyCommon.constructor()`.

### Recommendation

Consider adding below checks to ensure these input variables are not equal to `address(0)`:

```
require(_router != address(0), "Controller: _router is zero address");
router = _router;
require(_rewards != address(0), "Controller: _rewards is zero address");
rewards = _rewards;
```

## VHT-02 | Unlocked Compiler Version Declaration

Category	Severity	Location	Status
Language Specific	● Informational	gVaultHT.sol: 5, 166, 245, 388, 464, 490, 797, 819, 892, 907	ⓘ Acknowledged

### Description

The compiler version utilized throughout the project uses ' $\geq 0.6.12$ ', ' $\wedge 0.6.0$ ', ' $\wedge 0.6.12$ ', ' $\wedge 0.6.2$ ', denoting that a compiler version which is greater than the version will be used to compile the contracts.

### Recommendation

It is a general practice to instead lock the compiler at a specific version rather than allow a range of compiler versions to be utilized to avoid compiler-specific bugs and be able to identify ones more easily. We recommend locking the compiler at the lowest possible version that supports all the capabilities wished by the codebase. This will ensure that the project utilizes a compiler version that has been in use for the longest time and as such is less likely to contain yet-undiscovered bugs.

## VHT-03 | Missing Emit Events

Category	Severity	Location	Status
Gas Optimization	● Informational	gVaultHT.sol: 849(VaultCommon), 855(VaultCommon), 861(VaultCommon)	ⓘ Acknowledged

### Description

Several sensitive actions are defined without event declarations.

1. Functions `setController()`, `setStrategist()`, `setRouter()`, `setKeeper()`, `setGovernance()` can change the governance of the contracts.
2. Functions `setMin()`, `setSplit()`, `setWithdrawalFee()`, `setHarvestReward()`, `setStrategistReward()` will decide amounts that are critical to the project.

### Recommendation

Consider adding events for sensitive actions, and emit it in the function.

## VHT-04 | Proper Usage of “public” and “external” type

Category	Severity	Location	Status
Coding Style	● Informational	gVaultHT.sol: 855, 861, 1012	📄 Acknowledged

### Description

`public` functions that are never called by the contract could be declared `external`. When the inputs are arrays, `external` functions are more efficient than `public` functions.

For example: `setRewards()`, `setStrategist()`, `setSplit()`, `setRouter()`, `setGovernance()`, `setVault()`, `approveStrategy()`, `revokeStrategy()`, `setConverter()`, `setStrategyWithoutWithdraw()`, `setStrategy()`, `withdrawAll()`, `withdrawAllFromStrategy()`, `inCaseStrategyTokenGetStuck()`, `getExpectedReturn()`, `yearn()`, `withdraw()` in contract `Controller`. `setGovernance()`, `setController()` in contract `VaultCommon`. `getPricePerFullShare()` in contract `gVault`. `getPricePerFullShare()` in contract `gVaultHT`. `balanceOf()`, `withdraw()`, `harvest()` in contract `StrategyCommon`.

### Recommendation

Consider using the `external` attribute for functions never called from the contract.



## VHT-05 | No Restriction on Setting of Sensitive Variable

Category	Severity	Location	Status
Control Flow	● Minor	gVaultHT.sol: 851(VaultCommon)	ⓘ Acknowledged

### Description


The value of `min`, `withdrawalFee`, `strategistReward`, `harvestReward` can be arbitrarily updated by the `governance` role, which initially is the contract deployer. For example, `min` should always be less than or equal to the value of `max`.

### Recommendation

Consider to enforce any changes on the values of these variables to be restricted by Timelock, Multisig, or other DAO mechanism. Meanwhile, add following check on `min` to guarantee its value is always less than or equal to `max`.

```
require(_min <= max, "_min is over max");
```

## VHT-06 | Constrain Times of Function Calls

Category	Severity	Location	Status
Gas Optimization	Minor	gVaultHT.sol: 961(gVaultHT)	 Acknowledged

### Description

In `deposit()` function of `gVault` and `gVaultHT`, no matter how much the deposit amount is, the function will always call `earn()` method transferring token to strategy via controller. This process will consume lots of gas.

### Recommendation

Consider add a variable like `earnLowerlimit` as a threshold of the deposit amount, and corresponding setter function. `earn()` function will be called only if the balance of the vault reaches the value of `earnLowerlimit`.

For example:

```
if (token.balanceOf(address(this)) > earnLowerlimit){  
    earn();  
}
```

Additionally, the variable `min` equals to 10000 currently. Hence the `available()` will be 100% of the balance. In case user want to withdraw, the controller needs to withdraw from the strategy, which will waste gas.

```
uint256 public min = 10000;
```

Consider to set `min` to 9500.

# Appendix

## Finding Categories

### Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

### Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

### Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how `block.timestamp` works.

### Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

### Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

### Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a struct assignment operation affecting an in-memory struct rather than an in-storage one.

### Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete`.

### Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

## Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

# Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK's prior written consent.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

## About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.

