QuillAudits

Audit Report
July, 2021

YearnAgnostic

Contents

Introduction

During the period of **June 29th, 2021 to July 13th, 2021** – QuillHash Team performed a security audit for **Yearn Agnostic – Strategies and Bridge** smart contracts. The code for audit was taken from the following links:

https://github.com/Yearn-Agnostic/contracts/tree/develop/contracts/strategies

https://github.com/Yearn-Agnostic/contracts/tree/develop/contracts/bridge

Updated Code:
Commit Hash - a98681bad932207a63401f966758ec214724c177

Overview of Contract

Yearn Agnostic Finance supports decentralized finance (DeFi) projects deployed on Ethereum blockchain, Binance smart chain, Tron chain, etc., focusing on simplicity, user experience, security, and privacy. Yearn Agnostic Finance is a DeFi yield aggregator platform providing a classical way to optimally earn yield or maximize profits on assets. Yearn Agnostic Finance is a token-based ecosystem, and the underline token is YFIAG Token.

Benefits

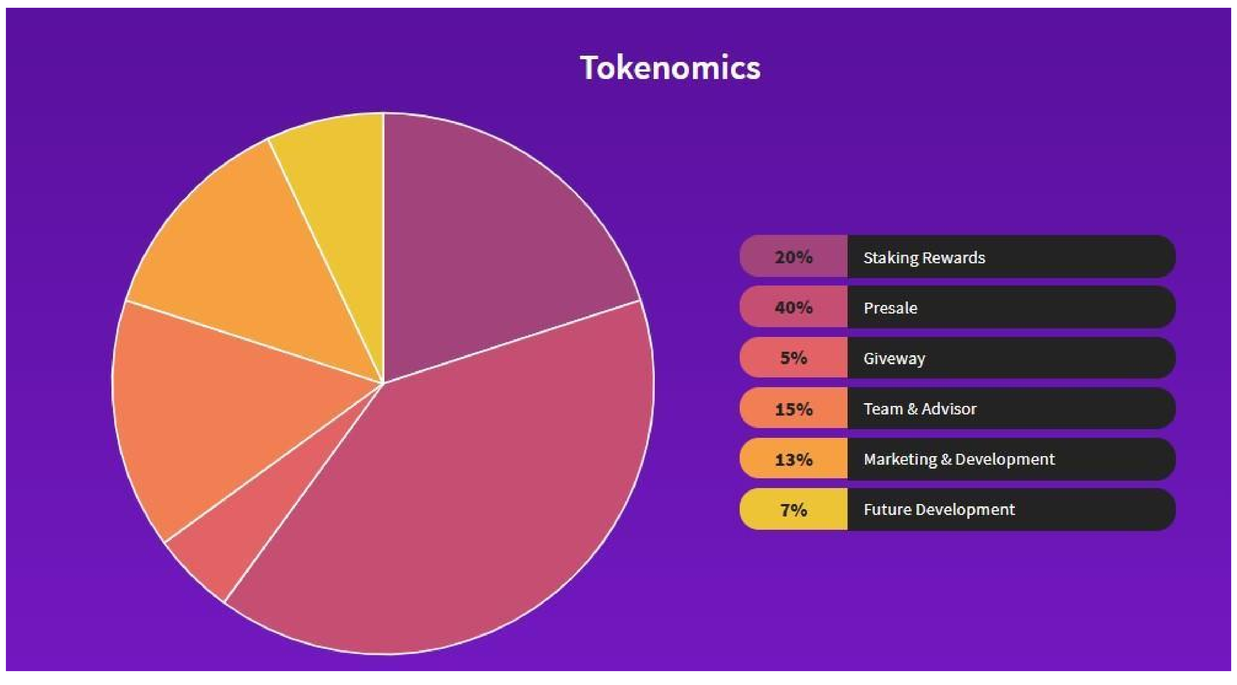
- Best way to earn an optimal interest rate on assets
- Simplifying earning process
- Community voting for fairness and equity
- Simple to use by everyone
- Privacy and anonymous transactions

Features

- LENDING
- STAKING
- BORROWING
- GOVERNANCE

Tokenomics

- 20% - Staking Rewards
- 40% - Presale
- 5% - Giveway
- 15% - Team & Advisor
- 13% - Marketing & Development
- 7% - Future Development



Details: https://yearnagnostic.finance/, https://yearnagnostic.io/

Scope of Audit

The scope of this audit was to analyze Yearn Agnostic – Strategies and Bridge smart contract codebase for quality, security, and correctness.
Following is the list of smart contracts included in the scope of this audit:

STRATEGIES

- BoilerplateStrategy.sol
- CompoundInteractor.sol
- CompoundNoFoldStrategy.sol
- CRVStrategyStable.sol
- CRVStrategyWRenBTCMix.sol
- CRVStrategyYCRV.sol
- VenusInteractor.sol
- VenusStrategy.sol

BRIDGE

- BaseBridge.sol
- ECDSA.sol
- BscBridge.sol
- EthBridge.sol

OUT-OF-SCOPE: External open-source contracts, external oracles, other smart contracts in the repository, or imported smart contracts.

Checked Vulnerabilities

We have scanned the smart contracts for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level
- Address hardcoded
- Using delete for arrays
- Integer overflow/
- underflow
- Locked money
- Private modifier
- Revert/require functions
- Using var
- Visibility
- Using blockhash
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

Techniques and Methods

Throughout the audit of the smart contracts care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

A combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy regarding the scope of the smart contract audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of smart contracts and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

Structural Analysis
In this step, we have analysed the design patterns and structure of smart contracts. A thorough check was done to ensure the Smart contract is structured in a way that will not result in future problems.

Code Review / Manual Analysis
Manual Analysis or review of code was done to identify new vulnerability or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Static Analysis

Static Analysis of smart contracts was done to identify contract vulnerabilities. In this step series of automated tools are used to test the security of smart contracts.

Gas Consumption

In this step, we have checked the behaviour of smart contract in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Remix IDE, Truffle, Truffle Team, Ganache, Solhint, Mythril, Manticore, Slither.

Issue Categories

High severity issues

Issues that must be fixed before deployment else they can create major issues.

Medium level severity issues

These issues will not create major issues in working but affect the performance of the smart contract.

Low level severity issues

These issues are more suggestions that should be implemented to refine the code in terms of gas, fees, speed and code accuracy

Number of issues per severity

Type	High	Medium	Low	Informational
Open	0	0	0	0
Acknowledged	0	4	0	3
Closed	0	5	7	0

Findings and Tech Details

STRATEGIES - BoilerplateStrategy.sol

1. Log the important events inside functions

   Code Lines: 63-65, 67-69
   Severity: Low
   It's a good practice to log important events. The functions like setController(address) and setStrategist(address) should emit events.

   Auditors Remarks: **Fixed**

2. **Coding Style Issues**

   Code Lines: PLEASE CHECK ALL THE FILES FOR CODING STYLE AND NAMING CONVENTION ISSUES.
   Severity: Informational
   Coding style issues influence code readability and, in some cases, may lead to bugs in the future. Smart Contracts have a naming convention, indentation, and code layout issues. It's recommended to use Solidity Style Guide to fix all the issues. Consider following the Solidity guidelines on formatting the code and commenting for all the files. It can improve the overall code quality and readability. The order of functions as well as the rest of the code layout does not follow solidity style guide. Layout contract elements in the following order:
   - Pragma statements
   - Import statements
   - Interfaces
   - Libraries
   - Contracts

   Inside each contract, library or interface, use the following order:
   - Type declarations
   - State variables
   - Events
   - Functions

   Auditors Remarks: **Acknowledged by the Auditee**
   Yearn Agnostic team acknowledged that it won't be having any impact on the contract.

STRATEGIES - CompoundInteractor.sol

None.

STRATEGIES - CompoundNoFoldStrategy.sol

1. Reentrancy - Avoid state changes after external calls

Severity: Medium
One of the major dangers of calling external contracts is that they can take over the control flow, and make changes to your data that the calling function wasn't expecting. It's recommended that the calls to external functions/events should happen after any changes to state variables in your contract so your contract is not vulnerable to a reentrancy exploit. When control is transferred to the recipient, care must be taken to not create reentrancy vulnerabilities. OpenZeppelin has its own mutex implementation you can use called ReentrancyGuard. This library provides a modifier you can apply to any function called nonReentrant that guards the function with a mutex. Consider using ReentrancyGuard or the checks-effects-interactions pattern.
Following link explains more about Reentrancy and ReentrancyGuard

- https://docs.openzeppelin.com/contracts/3.x/api/security#ReentrancyGuard
- https://blog.openzeppelin.com/reentrancy-after-istanbul/

Reentrancy safe example:

https://docs.soliditylang.org/en/v0.6.12/solidity-by-example.html?highlight=Reentrancy%20safe#safe-remote-purchase

Code Lines: State variables written/events emitted
constructor(address,address,address,address,address) [#17-33]
earn() [#126-133]
liquidateComp() [#140-165]
withdraw(uint256) [#71-97]
withdrawAll() [#102-112]

The function liquidateComp is not re-entrant safe. The restricted mode is not implemented in this function. Further ERC20 safeApprove has bugs. Here are more details:

- https://github.com/OpenZeppelin/openzeppelin-contracts/issues/2219
- https://drive.google.com/file/d/1skR4BpZ0VBSQICIC_eqRBgGnACf2li5X/view?usp=sharing
- https://docs.google.com/document/d/1YLPtQxZu1UAvO9cZ1O2RPXBbT0mooh4DYKjA_jp-RLM/edit
- https://blog.smartdec.net/erc20-approve-issue-in-simple-words-a41aaf47bca6
- https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729

Auditors Remarks: Acknowledged by the Auditee
Yearn Agnostic team acknowledged that they use a predefined set of public DeFi protocols with a predefined set of underlying tokens. Thus no unidentified external calls are performed.

2. **Log the important events inside functions**

   Code Lines: 64, 68, 90
   Severity: Low
   It's a good practice to log important events. The functions like setController(address), setStrategist(address) and setSellFloor(uint256) should emit events.

   Auditors Remarks: Fixed

STRATEGIES - CRVStrategyStable.sol

1. **Function convert has return type, but there is no return statement inside the function**

   Code Lines: 229
   Severity: Medium
   The function convert has return types, but the return statement is missing inside the function. It is recommended to add a return statement inside the function.

   Auditors Remarks: Fixed

2. **Log the important events inside functions**

   Code Lines: 64, 68
   Severity: Low
   It’s a good practice to log important events. The functions like setController(address) and setStrategist(address) should emit events.

   Auditors Remarks: **Fixed**

STRATEGIES - CRVStrategyWRenBTCMix.sol

1. **Reentrancy - Avoid state changes after external calls**

   Severity: Medium
   One of the major dangers of calling external contracts is that they can take over the control flow, and make changes to your data that the calling function wasn't expecting. It’s recommended that the calls to external functions/events should happen after any changes to state variables in your contract so your contract is not vulnerable to a reentrancy exploit. When control is transferred to the recipient, care must be taken to not create reentrancy vulnerabilities. OpenZeppelin has its own mutex implementation you can use called ReentrancyGuard. This library provides a modifier you can apply to any function called nonReentrant that guards the function with a mutex. Consider using ReentrancyGuard or the checks-effects-interactions pattern.
   Following link explains more about Reentrancy and ReentrancyGuard

   https://docs.openzeppelin.com/contracts/3.x/api/security#ReentrancyGuard

   https://blog.openzeppelin.com/reentrancy-after-istanbul/

   Reentrancy safe example:

   https://docs.soliditylang.org/en/v0.6.12/solidity-by-example.html?highlight=Reentrancy%20safe#safe-remote-purchase

   Code Lines: State variables written / events emitted claimAndLiquidateCrv() [#269-287]

The function liquidateComp is not re-entrant safe. The restricted mode is not implemented in this function. Further ERC20 safeApprove has bugs. Here are more details:

- https://github.com/OpenZeppelin/openzeppelin-contracts/issues/2219
- https://drive.google.com/file/d/1skR4BpZ0VBSQICIC_eqRBgGnACf2li5X/view?usp=sharing
- https://docs.google.com/document/d/1YLPtQxZu1UAvO9cZ1O2RPXBbT0mooh4DYKjA_jp-RLM/edit
- https://blog.smartdec.net/erc20-approve-issue-in-simple-words-a41aaf47bca6
- https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729

Auditors Remarks: Acknowledged by the Auditee
Yearn Agnostic team acknowledged that they use a predefined set of public DeFi protocols with a predefined set of underlying tokens. Thus no unidentified external calls are performed.

2. **Log the important events inside functions**

   Code Lines: 64, 68, 90
   Severity: Low
   It's a good practice to log important events. The functions like setController(address), setStrategist(address) and setSellFloor(uint256) should emit events.

   Auditors Remarks: Fixed

STRATEGIES - CRVStrategyYCRV.sol

1. **Log the important events inside functions**

   Code Lines: 64, 68, 90
   Severity: Low
   It's a good practice to log important events. The functions like setController(address), setStrategist(address) and setSellFloor(uint256) should emit events

   Auditors Remarks: Fixed

2. **Reentrancy - Avoid state changes after external calls**

   Severity: Medium

One of the major dangers of calling external contracts is that they can take over the control flow, and make changes to your data that the calling function wasn't expecting. It's recommended that the calls to external functions/events should happen after any changes to state variables in your contract so your contract is not vulnerable to a reentrancy exploit. When control is transferred to the recipient, care must be taken to not create reentrancy vulnerabilities. OpenZeppelin has its own mutex implementation you can use called ReentrancyGuard. This library provides a modifier you can apply to any function called nonReentrant that guards the function with a mutex. Consider using ReentrancyGuard or the checks-effects-interactions pattern.
Following link explains more about Reentrancy and ReentrancyGuard
https://docs.openzeppelin.com/contracts/3.x/api/security#ReentrancyGuard
https://blog.openzeppelin.com/reentrancy-after-istanbul/

Reentrancy safe example: https://docs.soliditylang.org/en/v0.6.12/solidity-by-example.html?highlight=Reentrancy%20safe#safe-remote-purchase

Code Lines: State variables written / events emitted claimAndLiquidateCrv() [#175-197]

The function claimAndLiquidateCrv() is not re-entrant safe. The restricted mode is not implemented in this function. Further ERC20 safeApprove has bugs. Here are more details:

- https://github.com/OpenZeppelin/openzeppelin-contracts/issues/2219
- https://drive.google.com/file/d/1skR4BpZ0VBSQICIC_eqRBgGnACf2li5X/view?usp=sharing
- https://docs.google.com/document/d/1YLPtQxZu1UAvO9cZ1O2RPXBbT0mooh4DYKjA_jp-RLM/edit
- https://blog.smartdec.net/erc20-approve-issue-in-simple-words-a41aaf47bca6
- https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729

Auditors Remarks: Acknowledged by the Auditee
Yearn Agnostic team acknowledged that they use a predefined set of public DeFi protocols with a predefined set of underlying tokens. Thus no unidentified external calls are performed.

STRATEGIES - VenusInteractor.sol

None.

STRATEGIES - VenusStrategy.sol

1. Reentrancy - Avoid state changes after external calls

Severity: Medium
One of the major dangers of calling external contracts is that they can take over the control flow, and make changes to your data that the calling function wasn't expecting. It's recommended that the calls to external functions/events should happen after any changes to state variables in your contract so your contract is not vulnerable to a reentrancy exploit. When control is transferred to the recipient, care must be taken to not create reentrancy vulnerabilities. OpenZeppelin has its own mutex implementation you can use called ReentrancyGuard. This library provides a modifier you can apply to any function called nonReentrant that guards the function with a mutex. Consider using ReentrancyGuard or the checks-effects-interactions pattern.
Following link explains more about Reentrancy and ReentrancyGuard

- https://docs.openzeppelin.com/contracts/3.x/api/security#ReentrancyGuard
- https://blog.openzeppelin.com/reentrancy-after-istanbul/

Reentrancy safe example:

https://docs.soliditylang.org/en/v0.6.12/solidity-by-example.html?highlight=Reentrancy%20safe#safe-remote-purchase

Code Lines: State variables written / events emitted
withdraw(uint256) [VenusStrategy.sol#69-94]
withdrawAll() [VenusStrategy.sol#99-109]
earn() [VenusStrategy.sol#121-128]
liquidateVenus() [VenusStrategy.sol#136-161]

The function liquidateVenus is not re-entrant safe. The restricted mode is not implemented in this function. Further, ERC20 safeApprove has bugs. Here are more details:

- https://github.com/OpenZeppelin/openzeppelin-contracts/issues/2219
- https://drive.google.com/file/d/1skR4BpZ0VBSQICIC_eqRBgGnACf2li5X/view?usp=sharing
- https://docs.google.com/document/d/1YLPtQxZu1UAvO9cZ1O2RPXBbT0mooh4DYKjA_jp-RLM/edit
- https://blog.smartdec.net/erc20-approve-issue-in-simple-words-a41aaf47bca6
- https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729

Auditors Remarks: Acknowledged by the Auditee
Yearn Agnostic team acknowledged that they use a predefined set of public DeFi protocols with a predefined set of underlying tokens. Thus no unidentified external calls are performed.

2. **Log the important events inside functions**

   Code Lines: 64, 68, 90
   Severity: Low
   It's a good practice to log important events. The functions like setController(address), setStrategist(address) and setSellFloor(uint256) should emit events

   Auditors Remarks: Fixed

BRIDGE - BaseBridge.sol

None.

BRIDGE - ECDSA.sol

1. **Function parseMessage has return type, but there is no return statement inside the function**

   Code Lines: 114
   Severity: Medium
   The function parseMessage has return types, but the return statement is missing inside the function. It is recommended to add a return statement inside the function.

   Auditors Remarks: Fixed

2. **Pure functions should not read/modify the state**

   Code Lines: 71, 114
   Severity: Medium
   Using inline assembly that contains certain opcodes is considered as reading/modifying the state. Functions that read/modify the state should not be declared as pure functions.
   Pure Functions- https://docs.soliditylang.org/en/v0.6.12/contracts.html#pure-functions
   EIP8854 - https://github.com/ethereum/solidity/issues/8854

   Auditors Remarks: Fixed

3. **Use external function modifier instead of public**

   Code Lines: 9-19
   Severity: Low
   The public functions that are never called by contract should be declared external to save gas. List of functions that can be declared external:
   - formMessage(address,address,uint256,uint256)
   These functions are never directly called by another function in the same contract or in any of its descendants. Consider marking them as "external" instead.

   Auditors Remarks: Fixed

4. **Consider using require instead of revert inside if..else**

   Code Lines: 76
   Severity: Informational
   In function recoverAddress() consider using require to check the condition. The two ways if (!condition) revert(...); and require(condition, ...); are equivalent. Use of require provides more readability.

   Auditors Remarks: Acknowledged by the Auditee
   Yearn Agnostic team acknowledged that it won't be having any impact on the contract.

5. **Potentially unbounded data structure**

   Code Lines: 54
   Severity: Informational
   Gas consumption in function "hashMessage" in contract "ECDSA" depends on the size of data structures that may grow unboundedly. Specifically the "1-st" argument to builtin "keccak256" may be able to grow unboundedly, causing the builtin to consume more gas than the block gas limit, effectively causing a denial-of-service condition. Consider that an attacker might attempt to cause this condition on purpose.

   Auditors Remarks: **Acknowledged by the Auditee**
   Yearn Agnostic team acknowledged that it won't be having any impact on the contract.

BRIDGE - BscBridge.sol

1. **Reentrancy - Avoid state changes after external calls**

   Severity: Medium
   One of the major dangers of calling external contracts is that they can take over the control flow, and make changes to your data that the calling function wasn't expecting. It's recommended that the calls to external functions/events should happen after any changes to state variables in your contract, so your contract is not vulnerable to a reentrancy exploit. When control is transferred to the recipient, care must be taken to not create reentrancy vulnerabilities. OpenZeppelin has its own mutex implementation you can use called ReentrancyGuard. This library provides a modifier you can apply to any function called nonReentrant that guards the function with a mutex.
   Consider using ReentrancyGuard or the checks-effects-interactions pattern.
   Following link explains more about Reentrancy and ReentrancyGuard
   https://docs.openzeppelin.com/contracts/3.x/api/security#ReentrancyGuard
   https://blog.openzeppelin.com/reentrancy-after-istanbul/

Reentrancy safe example:
https://docs.soliditylang.org/en/v0.6.12/solidity-by-example.html?highlight=Reentrancy%20safe#safe-remote-purchase

Code Lines: State variables written / events emitted
sendTokens(address,uint256,bytes) [#22-40]
unlockTokens(address,address,uint256,uint256,bytes) [#54-67]

Auditors Remarks: Fixed

BRIDGE - EthBridge.sol

1. **Reentrancy - Avoid state changes after external calls**

   Severity: Medium
   One of the major dangers of calling external contracts is that they can take over the control flow, and make changes to your data that the calling function wasn't expecting. It's recommended that the calls to external functions/events should happen after any changes to state variables in your contract, so your contract is not vulnerable to a reentrancy exploit. When control is transferred to the recipient, care must be taken to not create reentrancy vulnerabilities. OpenZeppelin has its own mutex implementation you can use called ReentrancyGuard. This library provides a modifier you can apply to any function called nonReentrant that guards the function with a mutex. Consider using ReentrancyGuard or the checks-effects-interactions pattern.
   Following link explains more about Reentrancy and ReentrancyGuard
   https://docs.openzeppelin.com/contracts/3.x/api/security#ReentrancyGuard
   https://blog.openzeppelin.com/reentrancy-after-istanbul/

   Reentrancy safe example: https://docs.soliditylang.org/en/v0.6.12/solidity-by-example.html?highlight=Reentrancy%20safe#safe-remote-purchase

   Code Lines: State variables written / events emitted
   sendTokens(address,uint256,bytes) [#20-40]
   unlockTokens(address,address,uint256,uint256,bytes) [#54-69]

   Auditors Remarks: Fixed

Strategies and Bridge Function List

The following is the list of functions tested and checked for vulnerabilities during audit

BoilerplateStrategy (IStrategy)
- [Public] <Constructor>
- [External] setController
- [External] setStrategist
- [External] setProfitSharing
- [External] setHarvestOnWithdraw
- [External] setLiquidationAllowed
- [External] setSellFloor
- [External] withdraw
- [Internal] _profitSharing
- [Public] <Constructor>
- [External] setController
- [External] setStrategist
- [External] setProfitSharing
- [External] setHarvestOnWithdraw

CompoundInternaleractor
- [Public] <Constructor>
- [Internal] _compoundSupply
- [Internal] _compoundRedeemUnderlying
- [Internal] _compoundRedeem
CompoundNoFoldStrategy (BoilerplateStrategy, CompoundInternaleractor)
- [Public] <Constructor>
- [External] getNameStrategy
- [External] want
- [External] balanceOf
- [Public] deposit
- [External] withdraw
- [External] withdrawAll
- [External] emergencyExit
- [Public] earn
- [Public] claimComp
- [Public] liquidateComp
- [External] convert
- [External] skim

CRVStrategyStable (IStrategy, BoilerplateStrategy)
- [Public] <Constructor>
- [External] getNameStrategy
- [External] want
- [Public] balanceOf
- [Public] balanceOfUnderlying
- [Public] deposit
- [Public] withdraw
- [External] withdrawAll
- [External] emergencyExit
- [Public] earn
- [Internal] yCurveFromUnderlying
- [Internal] _withdrawSome
- [Internal] _withdrawAll

CRVStrategyWRenBTCMix (IStrategy, BoilerplateStrategy)
- [Public] <Constructor>
- [External] getNameStrategy
- [External] want
- [External] balanceOf
- [Public] deposit
- [Public] withdraw
- [External] withdrawAll
- [External] emergencyExit
- [Public] earn
- [Internal] mixFromWBTC
- [Internal] investMixedCoin
- [Internal] withdrawMixTokens
- [Internal] investedUnderlyingBalance
- [Internal] wrapCoinAmount
- [Public] claimAndLiquidateCrv
- [Internal] calcLPAmount
- [External] convert
- [External] skim

CRVStrategyYCRV (IStrategy, BoilerplateStrategy)
- [Public] <Constructor>
- [External] getNameStrategy
- [External] want
- [Public] balanceOf
- [Public] deposit
- [Public] withdraw
- [External] withdrawAll
- [External] emergencyExit
- [Public] earn
- [External] convert
- [External] skim
- [Public] claimAndLiquidateCrv
- [Internal] withdrawYCrvFromPool
- [Internal] yCurveFromDai
- [Internal] _profitSharing

VenusInteractor
- [Public] <Constructor>
- [Internal] _venusSupplies
- [Internal] _venusRedeemUnderlying
- [Internal] _venusRedeem
VenusStrategy (BoilerplateStrategy, VenusInteractor)
- [Public] <Constructor>
- [External] getNameStrategy
- [External] want
- [External] balanceOf
- [Public] deposit
- [External] withdraw
- [External] withdrawAll
- [External] emergencyExit
- [Public] earn
- [Public] claimVenus
- [Public] liquidateVenus
- [External] convert
- [External] skim
BaseBridge (Ownable)
- [Public] <Constructor>
- [External] GetTransactionId

[Library] ECDSA
- [Public] isMessageValid
- [Public] formMessage
- [Private] getSigner
- [Internal] hashMessage
- [External] recoverAddress
- [Internal] parseMessage
BscBridge (BaseBridge)
- [Public] <Constructor>
- [External] sendTokens
- [External] unlockTokens
EthBridge (BaseBridge)
- [Public] <Constructor>
- [External] sendTokens
- [External] unlockTokens

Automated Testing

Slither

Slither is an open-source Solidity static analysis framework. This tool provides rich information about Ethereum smart contracts and has the critical properties. It runs a suite of vulnerability detectors, prints visual information about contract details, and provides an API to easily write custom analyses.

```
INFO:Printers:
Compiled with solc
Number of lines: 734 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 7 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 0
Number of informational issues: 17
Number of low issues: 7
Number of medium issues: 0
Number of high issues: 1
ERCs: ERC20

+----------------------+-------------+-------+----------------------+--------------+----------------------+
|         Name         | # functions | ERCS  |     ERC20 info       | Complex code |      Features        |
+----------------------+-------------+-------+----------------------+--------------+----------------------+
|       SafeMath       |      13     |       |                      |      No      |                      |
|        IERC20        |      6      | ERC20 |     No Minting       |      No      |                      |
|                      |             |       | Approve Race Cond.   |              |                      |
|                      |             |       |                      |              |                      |
|      SafeERC20       |      6      |       |                      |      No      |      Send ETH        |
|                      |             |       |                      |              | Tokens interaction   |
|       Address        |      11     |       |                      |      No      |      Send ETH        |
|                      |             |       |                      |              |    Delegatecall      |
|                      |             |       |                      |              |      Assembly        |
| BoilerplateStrategy  |      19     |       |                      |      No      | Tokens interaction   |
|        IVault        |      8      |       |                      |      No      |                      |
+----------------------+-------------+-------+----------------------+--------------+----------------------+
INFO:Slither:BoilerplateStrategy.sol analyzed (7 contracts)
```

```
INFO:Printers:
Compiled with solc
Number of lines: 780 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 8 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 0
Number of informational issues: 11
Number of low issues: 1
Number of medium issues: 2
Number of high issues: 0

ERCs: ERC20

+----------------------+-------------+-------+----------------------+--------------+----------------------+
|         Name         | # functions | ERCS  |     ERC20 info       | Complex code |      Features        |
+----------------------+-------------+-------+----------------------+--------------+----------------------+
|       SafeMath       |      13     |       |                      |      No      |                      |
|        IERC20        |      6      | ERC20 |     No Minting       |      No      |                      |
|                      |             |       | Approve Race Cond.   |              |                      |
|                      |             |       |                      |              |                      |
|      SafeERC20       |      6      |       |                      |      No      |      Send ETH        |
|                      |             |       |                      |              | Tokens interaction   |
|       Address        |      11     |       |                      |      No      |      Send ETH        |
|                      |             |       |                      |              |    Delegatecall      |
|                      |             |       |                      |              |      Assembly        |
|   CTokenInterface    |      26     |       |                      |      No      |                      |
|  CompoundInteractor  |      4      |       |                      |      No      | Tokens interaction   |
| ComptrollerInterface |      25     |       |                      |      No      |                      |
|  InterestRateModel   |      2      |       |                      |      No      |                      |
+----------------------+-------------+-------+----------------------+--------------+----------------------+
INFO:Slither:CompoundInteractor.sol analyzed (8 contracts)
```

```
INFO:Printers:
Compiled with solc
Number of lines: 1366 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 14 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 1
Number of informational issues: 22
Number of low issues: 14
Number of medium issues: 5
Number of high issues: 0

ERCs: ERC20

+-------------------------+-------------+-------+---------------------+--------------+---------------------+
|          Name           | # functions | ERCS  |      ERC20 info     | Complex code |       Features      |
+-------------------------+-------------+-------+---------------------+--------------+---------------------+
|        SafeMath         |      13     |       |                     |      No      |                     |
|         IERC20          |      6      | ERC20 |     No Minting      |      No      |                     |
|                         |             |       | Approve Race Cond.  |              |                     |
|                         |             |       |                     |              |                     |
|       SafeERC20         |      6      |       |                     |      No      |      Send ETH       |
|                         |             |       |                     |              | Tokens interaction  |
|        Address          |      11     |       |                     |      No      |      Send ETH       |
|                         |             |       |                     |              |    Delegatecall     |
|                         |             |       |                     |              |      Assembly       |
|    CTokenInterface      |      26     |       |                     |      No      |                     |
| CompoundNoFoldStrategy  |      36     |       |                     |      No      |      Send ETH       |
|                         |             |       |                     |              | Tokens interaction  |
|  ComptrollerInterface   |      25     |       |                     |      No      |                     |
|   IUniswapV2Router01    |      19     |       |                     |      No      |    Receive ETH      |
|   IUniswapV2Router02    |      24     |       |                     |      No      |    Receive ETH      |
|         IVault          |      8      |       |                     |      No      |                     |
|   InterestRateModel     |      2      |       |                     |      No      |                     |
+-------------------------+-------------+-------+---------------------+--------------+---------------------+
INFO:Slither:CompoundNoFoldStrategy.sol analyzed (14 contracts)
```

```
INFO:Printers:
Compiled with solc
Number of lines: 1270 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 13 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 2
Number of informational issues: 46
Number of low issues: 11
Number of medium issues: 1
Number of high issues: 0

ERCs: ERC20

+----------------------+-------------+-------+---------------------+--------------+---------------------+
|         Name         | # functions | ERCS  |      ERC20 info     | Complex code |       Features      |
+----------------------+-------------+-------+---------------------+--------------+---------------------+
|       SafeMath       |      13     |       |                     |      No      |                     |
|        IERC20        |      6      | ERC20 |     No Minting      |      No      |                     |
|                      |             |       | Approve Race Cond.  |              |                     |
|                      |             |       |                     |              |                     |
|     SafeERC20        |      6      |       |                     |      No      |      Send ETH       |
|                      |             |       |                     |              | Tokens interaction  |
|      Address         |      11     |       |                     |      No      |      Send ETH       |
|                      |             |       |                     |              |    Delegatecall     |
|                      |             |       |                     |              |      Assembly       |
| CRVStrategyStable    |      36     |       |                     |      No      |      Send ETH       |
|                      |             |       |                     |              | Tokens interaction  |
| ICurveFi_DepositY    |      10     |       |                     |      No      |                     |
|   ICurveFi_SwapY     |      7      |       |                     |      No      |                     |
| IUniswapV2Router01   |      19     |       |                     |      No      |    Receive ETH      |
| IUniswapV2Router02   |      24     |       |                     |      No      |    Receive ETH      |
|       IVault         |      8      |       |                     |      No      |                     |
|      IYERC20         |      4      |       |                     |      No      |                     |
+----------------------+-------------+-------+---------------------+--------------+---------------------+
INFO:Slither:CRVStrategyStable.sol analyzed (13 contracts)
```

```
INFO:Printers:
Compiled with solc
Number of lines: 1302 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 17 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 2
Number of informational issues: 55
Number of low issues: 13
Number of medium issues: 1
Number of high issues: 0

ERCs: ERC20

+---------------------+-------------+-------+----------------------+--------------+---------------------+
|        Name         | # functions | ERCS  |     ERC20 info       | Complex code |      Features       |
+---------------------+-------------+-------+----------------------+--------------+---------------------+
|      SafeMath       |      13     |       |                      |      No      |                     |
|       IERC20        |      6      | ERC20 |     No Minting       |      No      |                     |
|                     |             |       | Approve Race Cond.   |              |                     |
|     SafeERC20       |      6      |       |                      |      No      |      Send ETH       |
|                     |             |       |                      |              | Tokens interaction  |
|      Address        |      11     |       |                      |      No      |      Send ETH       |
|                     |             |       |                      |              |   Delegatecall      |
|                     |             |       |                      |              |      Assembly       |
|  CRVStrategyYCRV    |      34     |       |                      |      No      |      Send ETH       |
|                     |             |       |                      |              | Tokens interaction  |
| ICurveFi_DepositY   |      10     |       |                      |      No      |                     |
|   ICurveFi_Gauge    |      9      |       |                      |      No      |                     |
|       Gauge         |      4      |       |                      |      No      |                     |
|    VotingEscrow     |      4      |       |                      |      No      |                     |
|       Mintr         |      1      |       |                      |      No      |                     |
|  ICurveFi_Minter    |      5      |       |                      |      No      |                     |
| IUniswapV2Router01  |      19     |       |                      |      No      |    Receive ETH      |
| IUniswapV2Router02  |      24     |       |                      |      No      |    Receive ETH      |
|       IVault        |      8      |       |                      |      No      |                     |
|       IYERC20       |      4      |       |                      |      No      |                     |
+---------------------+-------------+-------+----------------------+--------------+---------------------+
INFO:Slither:CRVStrategyYCRV.sol analyzed (17 contracts)
```

```
INFO:Printers:
Compiled with solc
Number of lines: 796 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 9 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 0
Number of informational issues: 11
Number of low issues: 2
Number of medium issues: 1
Number of high issues: 0

ERCs: ERC20

+--------------------+-------------+-------+----------------------+--------------+---------------------+
|        Name        | # functions | ERCS  |     ERC20 info       | Complex code |      Features       |
+--------------------+-------------+-------+----------------------+--------------+---------------------+
|      SafeMath      |      13     |       |                      |      No      |                     |
|       IERC20       |      6      | ERC20 |     No Minting       |      No      |                     |
|                    |             |       | Approve Race Cond.   |              |                     |
|     SafeERC20      |      6      |       |                      |      No      |      Send ETH       |
|                    |             |       |                      |              | Tokens interaction  |
|      Address       |      11     |       |                      |      No      |      Send ETH       |
|                    |             |       |                      |              |   Delegatecall      |
|                    |             |       |                      |              |      Assembly       |
|    IComptroller    |      9      |       |                      |      No      |                     |
|      IVERC20       |      32     | ERC20 |     ∞ Minting        |      No      |                     |
|                    |             |       | Approve Race Cond.   |              |                     |
| InterestRateModel  |      2      |       |                      |      No      |                     |
|  VenusInteractor   |      4      |       |                      |      No      | Tokens interaction  |
+--------------------+-------------+-------+----------------------+--------------+---------------------+
INFO:Slither:VenusInteractor.sol analyzed (9 contracts)
```

```
INFO:Printers:
Compiled with solc
Number of lines: 1181 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 14 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 1
Number of informational issues: 17
Number of low issues: 14
Number of medium issues: 4
Number of high issues: 0

ERCs: ERC20

+--------------------+-------------+-------+----------------------+--------------+---------------------+
|        Name        | # functions | ERCS  |     ERC20 info       | Complex code |      Features       |
+--------------------+-------------+-------+----------------------+--------------+---------------------+
|      SafeMath      |      13     |       |                      |      No      |                     |
|       IERC20       |      6      | ERC20 |     No Minting       |      No      |                     |
|                    |             |       | Approve Race Cond.   |              |                     |
|                    |             |       |                      |              |                     |
|     SafeERC20      |      6      |       |                      |      No      |      Send ETH       |
|                    |             |       |                      |              | Tokens interaction  |
|      Address       |      11     |       |                      |      No      |      Send ETH       |
|                    |             |       |                      |              |    Delegatecall     |
|                    |             |       |                      |              |      Assembly       |
|    IComptroller    |      9      |       |                      |      No      |                     |
|   IPancakeRouter   |      8      |       |                      |      No      |    Receive ETH      |
|      IVERC20       |      32     | ERC20 |     ∞ Minting        |      No      |                     |
|                    |             |       | Approve Race Cond.   |              |                     |
|                    |             |       |                      |              |                     |
|       IVault       |      8      |       |                      |      No      |                     |
| InterestRateModel  |      2      |       |                      |      No      |                     |
|   VenusStrategy    |      36     |       |                      |      No      |      Send ETH       |
|                    |             |       |                      |              | Tokens interaction  |
+--------------------+-------------+-------+----------------------+--------------+---------------------+
INFO:Slither:VenusStrategy.sol analyzed (14 contracts)
```

```
INFO:Printers:
Compiled with solc
Number of lines: 131 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 1 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 1
Number of informational issues: 2
Number of low issues: 0
Number of medium issues: 0
Number of high issues: 0


+-------+-------------+------+------------+--------------+------------+
|  Name | # functions | ERCS | ERC20 info | Complex code |  Features  |
+-------+-------------+------+------------+--------------+------------+
| ECDSA |      6      |      |            |      No      | Ecrecover  |
|       |             |      |            |              |  Assembly  |
+-------+-------------+------+------------+--------------+------------+
INFO:Slither:ECDSA.sol analyzed (1 contracts)
```

Slither didn't raise any critical issue with smart contracts. The smart contracts were well tested, and all the minor issues that were raised have been documented in the report. Also, all other vulnerabilities of importance have already been covered in the Findings and Tech Details section of the report.

Mythril

Mythril is a security analysis tool for EVM bytecode. It detects security vulnerabilities in smart contracts built for Ethereum. It uses symbolic execution, SMT solving, and taint analysis to detect a variety of security vulnerabilities.

ECDSA.sol

CompoundInteractor.sol

CRVStrategyWRenBTCMix.sol

CRVStrategyYCRV.sol

Mythril did not detect any high severity issue. All the considerable issues raised by Mythril are already covered in the Findings and Tech Details section of this report.

Manticore

Manticore is a symbolic execution tool for the analysis of smart contracts and binaries. During Symbolic Execution / EVM bytecode security assessment did not detect any high severity issue. All the considerable issues are already covered in the Findings and Tech Details of this report.

Closing Summary

Overall, the smart contract code is extremely well documented, follows a high-quality software development standard, contains many utilities and automation scripts to support continuous deployment/ testing/ integration, and does NOT contain any obvious exploitation vectors that QuillHash was able to leverage within the timeframe of testing allotted. Overall, the smart contracts adhered to token standard guidelines. No critical or major vulnerabilities were found in the audit. **Several issues of less severity were found and reported during the audit.**

The outcome of this security audit is satisfactory; due to time and resource constraints, only testing and verification of essential properties was performed to achieve objectives and deliverables set in the scope. QuillHash recommends performing further testing to validate extended safety and correctness in context to the whole set of contracts.

It is highly recommended to fix all medium severity issues.

Disclaimer

QuillHash audit is not a security warranty, investment advice, or an endorsement of the Yearn Agnostic platform. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the Yearn Agnostic Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

QuillAudits

Canada, India, Singapore and Uniked Kingdom

audits.quillhash.com

audits@quillhash.com