Farmland Finance Protocol

Security Assessment

November 23th, 2020

For :
Farmland-Finance Team @ Farmland

By :
Owan Li @ CertiK
guilong.li@certik.org
Bryan Xu @ CertiK
buyun.xu@certik.org

Disclaimer

CertiK reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are 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 review.

CertiK Reports do 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.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing 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.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has indeed completed a round of auditing with the intention to increase the quality of the company/product's IT infrastructure and or source code.

Overview

Project Summary

Project Name	Farmland Finance Protocol
Description	a decentralized cross-chain platform for DeFi farming and profit distribution
Platform	Ethereum; Solidity
Codebase	GitHub Repository
Commit	3a715b074990b6e8f7310f449a19655eca03b9ee 14f7ff47e580f2fe4944f04d69b7e90858fb94d5 5a02bc46bbd8cd1ac174a2e1d4c2f0cb1a6a33bd f5b0915dae971fae6c8d7ffe764da51370954cf2

Audit Summary

Delivery Date	Nov. 23, 2020
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Nov. 10, 2020 - Nov. 13, 2020

Vulnerability Summary

Total Issues	9
Total Critical	0
Total Major	0
Total Minor	3
Total Informational	6

Executive Summary

This report has been prepared for Farmland Finance 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 Dynamic Analysis, Static Analysis, and Manual Review 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.

Documentation

The sources of truth regarding the operation of the contracts in scope were lackluster and are something we advise to be enriched to aid in the legibility of the codebase as well as project. To help aid our understanding of each contract's functionality we referred to in-line comments and naming conventions.

These were considered the specification, and when discrepancies arose with the actual code behaviour, we consulted with the Farmland team or reported an issue.

System Overview

Farmland Finance is a decentralized platform for DeFi farming and profit distribution.

The core components are the Controll,Strategy and Vault, which allow users to deposit their cross-chain digital assets.

These assets are transferred to a third party service Uniswap. In exchange, users are issued LP tokens that represent their claim on their deposits.

Users will be incentivized for earning staking rewards in Unipool, and saving gas by aggregation staking.

This protocol has an external dependency. All user deposits are immediately transferred to a third-party service (like Uniswap). The system should only be used if the service is appropriately trusted.

Review Notes

The scope for the Farmland-Finance protocol contains the vault,strategy and controll function.

Certain optimization steps that we pinpointed in the source code mostly referred to coding standards and inefficiencies, however 3 minor vulnerabilities were identified during our audit that solely concerns the specification.

Certain discrepancies between the expected specification and the implementation of it were identified and were relayed to the team, however they pose no type of vulnerability and concern an optional code path that was unaccounted for.

The project has adequate doumentation and specification outside of the source files, but the code comment coverage was minimal.

Recommendations

Overall, the codebase of the contracts should be refactored to assimilate the findings of this report, enforce linters and / or coding styles as well as correct any spelling errors and mistakes that appear throughout the code to achieve a high standard of code quality and security.

Findings

ID	Title	Type	Severity
EXH-01	Unlocked Compiler Version Declaration	Language Specific	Informational
EXH-02	Incorrect Naming Convention Utilization	Coding Style	Informational
EXH-03	Proper Usage of "public" and "external" type	Optimization	Informational
EXH-04	Check Zero Address	Optimization	Informational
EXH-05	Missing Emit Events	Optimization	Minor
EXH-06	Simplifying Existing Code	Optimization	Informational
EXH-07	Check Zero Address	Optimization	Minor
EXH-08	Redundant Codes	Coding Style	Informational
EXH-09	Duplicated Codes	Coding Style	Informational
EXH-10	DAO Needed	Optimization	Minor

Exhibit-01: Unlocked Compiler Version Declaration

Type	Severity	Location
Language Sepcific	Informational	Controller.sol, StrategyDForceDAI.sol, VaultDai.sol, StrategyCRV.sol,VaultRenbtc.sol, StrategyFortubeUsdc.sol,VaultUsdc.sol, StrategyDForceDAFortube.sol,VaultUsdt.sol

Description:

The compiler version utilized throughout the project uses the "^" prefix specifier, denoting that a compiler version which is greater than the version will be used to compile the contracts. Recommend the compiler version should be consistent throughout the codebase.

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.

 Exhibit-02: Incorrect Naming Convention Utilization

Type	Severity	Location
Coding Style	Informational	StrategyDForceDAI.sol L124,L132,L149,L151,L153 Controller.sol L188 StrategyCRV.sol L171,L176 StrategyFortubeUsdc.sol L161,L164 StrategyFortube.sol L161,L164

Description:

Solidity defines a naming convention that should be followed. In general, parameters should use mixedCase, refer to: https://solidity.readthedocs.io/en/v0.5.16/style-guide.html#naming-conventions

Functions other than constructors should use mixedCase.
Examples:
Contract like: `dRewards`, `dERC20`

Constants should use UPPER_CASE_WITH_UNDERSCORES.
Examples:
Constants like: `d`, `df`, `unirouter`, `curveminter`, `curvedeposit`

Parameter shoud use mixedCase.
Examples:
Parameter like: `_onesplit`

Recommendation:

The recommendations outlined here are intended to improve the readability, and thus they are not rules, but rather guidelines to try and help convey the most information through the names of things.

Exhibit-03: Proper Usage of "public" and "external" type

Type	Severity	Location
Optimization	Informational	VaultDai.sol L465,Controller.sol L244,L256, VaultRenbtc.sol L391,L412,L456, VaultUsdc.sol L465,VaultUsdt.sol L465

Description:

"public" functions that are never called by the contract should be declared "external" . When the inputs are arrays "external" functions are more efficient than "public" functions.
Examples
Functions like : `getPricePerFullShare()`
, `inCaseTokensGetStuck`, `yearn()`, `batchDepositbtc`, `balanceOfBtc`

Recommendation:

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

Exhibit-04: Check Zero Address

Type	Severity	Location
Optimization	Informational	Controller.sol L193, StrategyDForceDAI.sol L340, VaultDai.sol L384,VaultRenbtc.sol L318, StrategyCRV.sol L340, StrategyFortubeUsdc.sol L320, VaultUsdc.sol L384, StrategyFortube.sol L319, VaultUsdt.sol L384

Description:

The function `setGovernance` does not verified the address before usage.

Recommendation:

We recommend adding below code:
`require(_governance != address(0), "Address zero is forbbiden");`
（Farmland - Resolved） The issue is addressed in commit 14f7ff47e580f2fe4944f04d69b7e90858fb94d5.

Exhibit-05: Missing Emit Events

Type	Severity	Location
Optimization	Minor	Controller.sol, StrategyDForceDAI.sol, VaultDai.sol, StrategyCRV.sol,VaultRenbtc.sol, StrategyFortubeUsdc.sol,VaultUsdc.sol, StrategyDForceDAFortube.sol,VaultUsdt.sol

Description:

Several sensitive actions are defined without event declarations.

1. Fucntion `setGovernance()` can change the governance of the contracts.
2. Fucntions `setMin()`, `available()` will decide the proportion of assest to be borrowed.
3. Function `inCaseTokensGetStuck()` can transfer any amount of assets to governance in case the controller has.
4. Fucntions `setFee()`,`setCallFee()`,`setWithdrawalFee()` will decide the important metrics.

Recommendation:

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

(Farmland - Resolved) The issue is addressed in commit f5b0915dae971fae6c8d7ffe764da51370954cf2.

Exhibit-06: Simplifying Existing Code

Type	Severity	Location
Optimization	Informational	Controller.sol L178,L183,L188,L193,L198,L205,L210,L239,L244 StrategyDForceDAI.sol L340,L345,L349,L353,L358,L364 VaultDai.sol L379,L384,L389,L393 StrategyCRV.sol L349,L345,L350,L354,L358,VaultRenbtc.sol L313,L318,L323,L327, StrategyFortubeUsdc.sol L320,L325,L329,L333,L338,L344 VaultUsdc.sol L379, L384,L389,L393 StrategyDForceDAFortube.sol L319,L324,L328,L332,L337,L348,VaultUsdt.sol L379, L384,L389,L393

Description:

Consider using a modifier to replace the below same codes existing in many functions:
require(msg.sender == governance, "!governance");
Example:

Functions `setFactory()`, `setSplit()`, `setOneSplit()`,`setGovernance()`,`setVault()`,`setConverter()`,`setStrategy()`,`withdrawAll()`,`inCaseTokensGetStuck()` in Controller.sol.
Functions `setGovernance()`,`setController()`,`setFee()`,`setCallFee()`,`setWithdrawalFee()`,`setLastHarvestTimestampSec()` in StrategyDForceDAI.sol.

Functions `setMin()`, `setGovernance()`, `setController()`, `setEarnLowerlimit()` in VaultDai.sol

Recommendation:

Consider changing it as following example:

```
modifier onlyGovernance() {
        require(msg.sender == governance, "!governance");
        _;
    }
```

（Farmland - Resolved） The issue is addressed in commit 14f7ff47e580f2fe4944f04d69b7e90858fb94d5.

Exhibit-07: Check Zero Address

Type	Severity	Location
Optimization	Minor	Controller.sol L215

Description:

Function `earn()` is public, anyone can call it with any parameters.
The address of `_strategy` is not checked. If it's a zero address, assets in `Controller` will be transferred to zero address.

Recommendation:

We recommend adding below code:

```
require(_strategy != address(0), "Strategy is not exist!");
```

Exhibit-08: Redundant Codes

Type	Severity	Location
Coding Style	Informational	Controller.sol L160,L178

Description:

Variable `factory` and function `setFactory()` are defined but never used.

Recommendation:

We recommend removing the redundant codes.
(Farmland - Resolved) The issue is addressed in commit 14f7ff47e580f2fe4944f04d69b7e90858fb94d5.

Exhibit-09: Duplicated codes

Type	Severity	Location
Optimization	Informational	VaultRenbtc.sol L371,L395

Description:

There are below duplicated codes in both `withdraw` and `_checkBtcWithdrawal` :

```
    uint r = (balance().mul(_shares)).div(totalSupply());
    _burn(msg.sender, _shares);

    // Check balance
    uint b = token.balanceOf(address(this));
    if (b < r) {
        uint _withdraw = r.sub(b);
        Controller(controller).withdraw(address(token), _withdraw);
        uint _after = token.balanceOf(address(this));
        uint _diff = _after.sub(b);
        if (_diff < _withdraw) {
            r = b.add(_diff);
        }
    }
```

Recommendation:

We recommend to move the duplicated codes out of the code block and they can be replaced by one.
(Farmland - Resolved) The issue is addressed in commit 14f7ff47e580f2fe4944f04d69b7e90858fb94d5.

Exhibit-10: DAO needed

Type	Severity	Location
Optimization	Minor	StrategyDForceDAI.sol L247, StrategyCRV.sol L267, StrategyFortubeUsdc.sol L254, StrategyFortube.sol L255

According to the code and doc, the Farmland DAO and Timelock is not completed yet. Governance ownership is better to be transferred to Timelock or Farmland DAO when it is implemented.
See below sample code, function withdrawAll in StrategyDForceDAI.sol.

```
    function withdrawAll() external returns (uint balance) {

        require(msg.sender == controller, "!controller");
        _withdrawAll();


        balance = IERC20(want).balanceOf(address(this));

        address _vault = Controller(controller).vaults(address(want));
        require(_vault != address(0), "!vault"); // additional protection so we
don't burn the funds
        IERC20(want).safeTransfer(_vault, balance);
    }
```

This is same code as yfii
StrategyDForce.sol. But it is possible to avoid the check by setting controller to a new address. Then the balance can be transferred to a fake vault. The same codes are in StrategyCRV.sol, StrategyFortubeUsdc.sol, StrategyFortube.sol.

Recommendation:

We recommend to use multi-sig wallet as governor to ensure the security of user assets. Consider adding an event for `setController()`, and emit it in the function.

(Farmland - Response) Governance ownership transferring to the Timelock is under planning now.