Deerfi Protocol

Security Assessment

November 23th, 2020

For :
Deerfi Team @ Deerfi

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

Disclaimer

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	Deerfi Protocol
Description	a decentralized money market that offers users access to permissionless lending and borrowing
Platform	Ethereum; Solidity
Codebase	GitHub Repository
Commit	60f2284cf11bd5eb9df8205aed37e3344b166f0e cb4dd660b54280ee9f3670c99ce281fc41ed9c1f dcb84ff6302acfc8e1666d0faf2f826d704d0e6e

Audit Summary

Delivery Date	Nov. 23, 2020
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Nov. 18, 2020 - Nov. 20, 2020

Vulnerability Summary

Total Issues	7
Total Critical	0
Total Major	0
Total Minor	0
Total Informational	7

Executive Summary

This report has been prepared for Deerfi Portocol 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 Deerfi team or reported an issue.

Review Notes

The Deerfi protocol is a decentralized money market , offerring users lending/borrowing functionalities. The Deerfi Protocol is a fork of Compound Finance that is powered by Chainlink oracles. The audited commits are 60f2284cf11bd5eb9df8205aed37e3344b166f0e, cb4dd660b54280ee9f3670c99ce281fc41ed9c1f, dcb84ff6302acfc8e1666d0faf2f826d704d0e6e and the files included in the scope were ChainlinkPriceOracleProxy.sol, IUniswapV2Pair.sol, Math.sol.
Compound Finance was audited by OpenZeppelin.

Certain optimization steps that we pinpointed in the source code mostly referred to coding standards and inefficiencies.

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.

There are several naming convention issues revealed in this audit, however Compound has the same naming convention issues. To be consistent with Compound, these issues will not be resolved currently.

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	Gas Consumption	Optimization	Informational
EXH-05	State Variable Shadowing from Abstract Contracts	Optimization	Informational
EXH-06	Events Should Add Indexed Keyword	Optimization	Informational
EXH-07	Missing Emit Events	Optimization	Informational

Exhibit-01: Unlocked Compiler Version Declaration

Type	Severity	Location
Language Sepcific	Informational	ChainlinkPriceOracleProxy.sol, IUniswapV2Pair.sol

Description:

Different versions of Solidity is used in the project. The compiler version utilized in most files uses the “^0.5.16” prefix specifier, but “>=0.5.0” is used in `IUniswapV2Pair.sol`. 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.

(Deerfi - Response) IUniswapV2Pair.sol is directly reused from Uniswap-v2-core, prefer not modify it since it doesn’t affect security.

Exhibit-02: Incorrect Naming Convention Utilization

Type	Severity	Location
Coding Style	Informational	ChainlinkPriceOracleProxy.sol, IUniswapV2Pair.sol

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

Constants should use UPPER_CASE_WITH_UNDERSCORES.
Examples:
Constants like: `isPriceOracle`

Functions shoud use mixedCase.
Examples:
Functions like: `DOMAIN_SEPARATOR` , `PERMIT_TYPEHASH` , `MINIMUM_LIQUIDITY`

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

Events definition should be in front of function definitions:

```
event TokenConfigUpdated(
      address  cTokenAddress,
      address  chainlinkAggregatorAddress,
      uint256  chainlinkPriceBase,
      uint256  underlyingTokenDecimals
  );
```

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.

However Compound has the same naming convention issues. To be consistent with Compound, these issues will not be resolved currently.

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

Type	Severity	Location
Optimization	Informational	ChainlinkPriceOracleProxy.sol L64,L83,L92,L221

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 : `owner()` , `renounceOwnership`, `transferOwnership()`, `getUnderlyingPrice`

Recommendation:

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

(Deerfi - Response) We will change getUnderlyingPrice() to external. For the rest, we prefer to keep it as it is since that is how Ownable is defined.

(Deerfi - Resolved) The issue is addressed in commit d6952d04b6552413b950fd756e9cbffe8de735a6.

Exhibit-04: Gas Consumption

Type	Severity	Location
Optimization	Informational	ChainlinkPriceOracleProxy.sol L221

Description:

The function `getUnderlyingPrice()` does not verified the parameter `cToken` before usage.

Recommendation:

We recommend adding below code:

```
address cTokenAddress = address(cToken);
TokenConfig memory config = tokenConfig[cTokenAddress];
require(config.chainlinkPriceBase != 0, "Invalid config");
```

(Deerfi - Resolved) The issue is addressed in commit d6952d04b6552413b950fd756e9cbffe8de735a6.

Exhibit-05: State Variable Shadowing from Abstract Contracts

Type	Severity	Location
Optimization	Informational	ChainlinkPriceOracleProxy.sol L138,L157

Description:

State variable `isPriceOracle` in contract `ChainlinkPriceOracleProxy` is shadowed from abstract contract `PriceOracle`.

```
bool public constant isPriceOracle = true;
```

Recommendation:

We recommend to remove the state variable shadowing.

(Deerfi - Resolved) The issue is addressed in commit d6952d04b6552413b950fd756e9cbffe8de735a6.

Exhibit-06: Events Should Add Indexed Keyword

Type	Severity	Location
Optimization	Informational	ChainlinkPriceOracleProxy.sol L138,L157

Description:

Event definittions in contract `ChainlinkPriceOracleProxy` do not have `indexed` keyword. The indexed parameters for logged events will allow you to search for these events using the indexed parameters as filters.

```
event TokenConfigUpdated(
      address cTokenAddress,
      address chainlinkAggregatorAddress,
      uint256 chainlinkPriceBase,
      uint256 underlyingTokenDecimals
  );
```

Recommendation:

We recommend to add the `indexed` keyword.

```
event TokenConfigUpdated(
      address indexed cTokenAddress,
      address indexed chainlinkAggregatorAddress,
      uint256 chainlinkPriceBase,
      uint256 underlyingTokenDecimals
  );
```

(Deerfi - Resolved) The issue is addressed in commit d6952d04b6552413b950fd756e9cbffe8de735a6.

Exhibit-07: Missing Emit Events

Type	Severity	Location
Optimization	Informational	ChainlinkPriceOracleProxysol L248

Description:

Several sensitive actions are defined without event declarations.
Fucntion `setEthUsdChainlinkAggregatorAddress()` can change the chainlink price oracle address.

Recommendation:

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

(Deerfi - Response) We don’t have a plan to track on this event.