CERTIK

FireDAO Protocol

Security Assessment

Feburary 6th, 2021

For :
FireDAO Protocol

By :

Connie Lam @ CertiK
connie.lam@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	FireDAO Protocol
Description	An easy-to-use tool that maximizes returns for cryptocurrency assets by automatically deploying them to decentralized finance (DeFi) protocols that generate the highest yield via lending, farming and exchange services.
Platform	Ethereum; Solidity
Codebase	GitHub Repository
Checksum	3a715b074990b6e8f7310f449a19655eca03b9ee

Audit Summary

Delivery Date	Feb. 6, 2020
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Jan. 31, 2020 - Feb. 6, 2020

Vulnerability Summary

Total Issues	9
Total Critical	0
Total Major	0
Total Minor	2
Total Informational	7

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.

File in Scope

ID	Contract	SHA-256 Checksum
RWP	RewardPool.sol	55676ae355f4c11759536aa5247257eb3cbe88ee7680c6ecb63a645a2c5f69d1
CTL	Controllers.sol	f3f12b225fd8aa5a4e1ad3a76d054449a0dc46c42610865962783ea4293ebaf7
FIRE	FIRE.sol	d74f60b2f4aee5baa77bbb88283573ca526c28a97153aa442bfcae7753cb64d1
GOVALP	GovernorAlpha.sol	b3686ed316eca47b1acf78c415bc7f05937024d9ed7bd2356d931933008c44e2
RWD	Rewarder.sol	e51dbe54b4b0ba984ae507b4f222cd463f7185453e847c48f4daebbf53db0342
TIM	Timelock.sol	2220f49e740a02dcff704b65785e20a710510e445e1b55fcde6eab8ec15373c4
SDAI	StrategyDForceDAI.sol	4c3aac929b41d1dc2f7ab75ad8fa99c895256d1e5d5e9586144260b70da02806
SUSDC	StrategyDForceUSDC.sol	115a48e81a7dd0c04bd1eacb7c9db2d851d93b6f9aa2e01cd91d382012330508
SUSDT	StrategyDForceUSDT.sol	49c5bb69670b21e2620b82bf8d0e7a0368753258dc1aa55856a4aff049efe710
FVT	fiVault.sol	88839660dbef1355a52e3d3a875e4f58502aab31ef119009dcca2e5d3633cde7

Findings

ID	Title	Type	Severity
RWP-01	Unlocked Compiler Version	Language Specific	Informational
CTL-01	Unlocked Compiler Version	Language Specific	Informational
CTL-02	Missing Emit Events	Optimization	Minor
CTL-03	Simplifying Existing Code	Optimization	Informational
FIRE-01	Check Zero Address	Optimization	Minor
FIRE-02	Constant State Variable	Coding Style	Informational
SDAI-01	Unlocked Compiler Version	Language Specific	Informational
SUSDC-01	Unlocked Compiler Version	Language Specific	Informational
SUSDT-01	Unlocked Compiler Version	Language Specific	Informational
RWD-01	Unlocked Compiler Version	Language Specific	Informational
RWD-02	Lacks input validation	Volatile Code	Minor
FVT-01	Unlocked Compiler Version	Language Specific	Informational
FVT-02	Missing Emit Events	Optimization	Minor
FVT-03	Simplifying Existing Code	Optimization	Informational

RWP-01: Unlocked Compiler Version

Type	Severity	Location
Language Sepcific	Informational	RewardPool.sol

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version v0.6.2 the contract should contain the following line:

```
pragma solidity 0.6.2;
```

CTL-01: Unlocked Compiler Version

Type	Severity	Location
Language Sepcific	Informational	Controller.sol

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version v0.6.2 the contract should contain the following line:

```
pragma solidity 0.6.2;
```

CTL-02: Missing Emit Events

Type	Severity	Location
Optimization	Minor	Contoller.sol

Description:

Several sensitive actions are defined without event declarations.

Examples:
Functions like : `setRewards()` , `setStrategist()` , `setSplit()` , `setOneSplit()` , `setGovernance()` , `setVault()` , `approveStrategy()` , `revokeStrategy()` , `setConverter()` , `setStrategy()` in `Controller` contract;

Recommendation:

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

```
event setRewards(address indexed _rewards);
function setRewards(address _rewards) external {
  ...
  emit setRewards(_rewards);
}
```

CTL-03: Simplifying Existing Code

Type	Severity	Location
Optimization	Informational	Controller.sol L43,L48,L53,L58,L74,L79

Description:

Consider using a modifier to replace the below same codes existing in many functions:

```
    require(msg.sender == governance, "!governance");

```

Recommendation:

Consider changing it as following example:

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

FIRE-01: Check Zero Address

Type	Severity	Location
Optimization	Informational	FIRE.sol L100

Description:

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

Recommendation:

We recommend adding below code:

```
require(_minter != address(0), "FIRE::mint: _minter address cannot be the zero address");
```

FIRE-02: Constant State Variable

Type	Severity	Location
Coding Style	Informational	FIRE.sol L100

Description:

State variables `cap` can be declared as `constant`

Recommendation:

We recommend line 34 change to:
`uint256 public constant CAP = 100_000_000e18;`

SDAI-01: Unlocked Compiler Version

Type	Severity	Location
Language Sepcific	Informational	StrategyDForceDAI.sol

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version v0.6.2 the contract should contain the following line:

```
pragma solidity 0.6.2;
```

SUSDC-01: Unlocked Compiler Version

Type	Severity	Location
Language Sepcific	Informational	StrategyDForceUSDC.sol

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version v0.6.2 the contract should contain the following line:

```
pragma solidity 0.6.2;
```

SUSDT-01: Unlocked Compiler Version

Type	Severity	Location
Language Sepcific	Informational	StrategyDForceUSDT.sol

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version v0.6.2 the contract should contain the following line:

```
pragma solidity 0.6.2;
```

RWD-01: Unlocked Compiler Version

Type	Severity	Location
Language Sepcific	Informational	Rewarder.sol

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version v0.6.2 the contract should contain the following line:

```
pragma solidity 0.6.2;
```

RWD-02: Lacks input validation

Type	Severity	Location
Volatile Code	Informational	Rewarder.sol

Description:

Function doesn’t checks provided address array length.

Recommendation:

Consider checking the `address[] calldata recipients` addresses length are equal to `uint256[] calldata values`

```
require(recipients.length == values.length,
    "Rewarder::reward: reward function information arity mismatch")

```

FVT-01: Unlocked Compiler Version

Type	Severity	Location
Language Sepcific	Informational	fiVault.sol

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version v0.6.2 the contract should contain the following line:

```
pragma solidity 0.6.2;
```

FVT-02: Missing Emit Events

Type	Severity	Location
Optimization	Minor	fiVault.sol

Description:

Several sensitive actions are defined without event declarations.

Examples:
Functions like : `setMin()` , `setCap()` , `setGovernance()` , `setController()` in `fiVault` contract

Recommendation:

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

```
        event setGovernance(address indexed _rewards);
        function setGovernance(address _governance) external {
          ...
          emit setGovernance(_governance);
        }
```

FVT-03: Simplifying Existing Code

Type	Severity	Location
Optimization	Informational	fiVault.sol L49,L54,L59,L64

Description:

Consider using a modifier to replace the below same codes existing in many functions:

```
    require(msg.sender == governance, "!governance");

```

Recommendation:

Consider changing it as following example:

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

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that 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.

Mathematical Operations

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings are exhibits that 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 instorage 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 and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a `constructor` assignment imposing different `require` statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as `constant` contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.

Icons explanation

✓ : Issue resolved

ⓘ : Issue not resolved / Acknowledged. The team will be fixing the issues in the own timeframe.

ⓘ✓ : Issue partially resolved. Not all instances of an issue was resolved.