Bifrost Finance

Security Assessment

November 27th, 2020

For :
Bifrost Finance @ Bifrost

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	bifrost-finance
Description	An ERC20 token and Mint Drop Contract
Platform	Ethereum; Solidity
Codebase	bifrost-mint-drop
Commit	52e2be1422fc69400f91ea0700f13803dd75de66 d42fd40673bffe61b738459113b11cc3efd91392 41585a78c93f10a7289a1a3bfe0f6541443f0d40

Audit Summary

Delivery Date	Nov. 27, 2020
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Nov. 25, 2020 - Nov. 27, 2020

Vulnerability Summary

Total Issues	7
Total Critical	1
Total Major	0
Total Minor	1
Total Informational	5

Executive Summary

This report has been prepared for **Bifrost 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 **Bifrost Finance** or reported an issue.

Review Notes

The audited commits are 52e2be1422fc69400f91ea0700f13803dd75de66 , d42fd40673bffe61b738459113b11cc3efd91392, 41585a78c93f10a7289a1a3bfe0f6541443f0d40

and the files included in the scope are MintDrop.sol,vETH.sol and IVETH.sol.

Certain optimization steps that we pinpointed in the source code mostly referred to coding standards and inefficiencies, however 1 critial and 1 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.

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	Optimization	Informational
EXH-02	Proper Usage of "public" and "external" type	Coding Style	Informational
EXH-03	Gas Consumption	Optimization	Informational
EXH-04	Gas Consumption	Optimization	Informational
EXH-05	Dangerous require	Optimization	Critical
EXH-06	Code optimization	Optimization	Informational
EXH-07	Incorrect Rewards Calculation	Optimization	Minor

Exhibit-01: Unlocked Compiler Version Declaration

Type	Severity	Location
Language Sepcific	Informational	IVETH.sol

Description:

The compiler version utilized in several files uses the "^" prefix specifier, denoting that a compiler version which is greater than the version will be used to compile the contracts.
The compiler version utilized in other files uses the "<=" prefix specifier, denoting that a compiler version which is smaller 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.

(**Biforst - Resolved**) The issue is addressed in commit d42fd40673bffe61b738459113b11cc3efd91392.

Exhibit-02: Proper Usage of "public" and "external" type

Type	Severity	Location
Coding Style	Informational	MintDrop.sol

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. link
Examples:
Functions like : `getRewards()`

Recommendation:

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

(**Biforst - Resolved**) The issue is addressed in commit d42fd40673bffe61b738459113b11cc3efd91392.

Exhibit-03: Gas consumption

Type	Severity	Location
Optimization	Informational	MintDrop.sol L37,MintDrop.sol L39

Description:

Below variables change only once, better to define it as immutable to avoid gas consumption.

```
address public vETHAddress;
address public depositAddress;
uint public bonusStartAt;
```

Recommendation:

We recommend to change the codes as below:

```
address public immutable vETHAddress;
address public immutable depositAddress;
uint public immutable bonusStartAt;
```

(**Biforst - Resolved**) The issue is addressed in commit d42fd40673bffe61b738459113b11cc3efd91392.

Exhibit-04: Gas consumption

Type	Severity	Location
Optimization	Informational	MintDrop.sol L139

Description:

The function `lockWithdraw()` could add the modifier `isWithdrawNotLocked` to save gas in case of multiple calls.

Recommendation:

We recommend to add the modifier `isWithdrawNotLocked`.

```
function lockWithdraw() external onlyOwner isWithdrawNotLocked {}
```

(**Biforst - Resolved**) The issue is addressed in commit d42fd40673bffe61b738459113b11cc3efd91392.

Exhibit-05: Dangerous require

Type	Severity	Location
Optimization	Critical	MintDrop.sol L139

Description:

Because of the condition at line 123,the function `lockForValidator()` may never succeed.

Neither contracts nor “external accounts” are currently able to prevent that someone sends them Ether. Contracts can react on and reject a regular transfer, but there are ways to move Ether without creating a message call. One way is to simply “mine to” the contract address and the second way is using `selfdestruct(x)`.link

Recommendation:

We recommend to change it like this or simply delete it.

```
require(address(this).balance.add(totalLocked) >= totalDeposit, "invalid
balance");
```

(**Biforst - Resolved**) The issue is addressed in commit d42fd40673bffe61b738459113b11cc3efd91392.

Exhibit-06: Code optimization

Type	Severity	Location
Optimization	Informational	MintDrop.sol L96

Description:

Since the reward is calculated after the start time, it is better to compare it to the `bonusStartAt` instead of comparing with `0`.

Recommendation:

We recommend to change it like this.

```
        if (now < bonusStartAt) {
            if (myLastClaimedAt[msg.sender] < bonusStartAt) {
                myLastClaimedAt[msg.sender] = bonusStartAt;
            }
            return;
        }
        if (myLastClaimedAt[msg.sender] >= bonusStartAt) {}
```

(**Biforst - Resolved**) The issue is addressed in commit d42fd40673bffe61b738459113b11cc3efd91392.

Exhibit-07: Incorrect Rewards Calculation

Type	Severity	Location
Optimization	Minor	MintDrop.sol

Description:

In `MintDrop` contract, users must consistently call the function `claimRewards()` . Otherwise the `rewards` is incorrectly calculated.

Example:

User A deposited 1 ether on the first day, no other users deposited until the end of the 7th day.

If user A claim rewards at the end of the 7th day, he will get 100% of the total rewards for 7 days.

If user A forgot to claim rewards, and in the 8th day user B deposited 99 ethers.

At the end of the 8th day, user A claimed rewards, and he will only get down to 1% of the total rewards for all 8 days.

(**Biforst - Response**) This rewards calculation is not entirely fair, but it is relatively simple to calculate.