PolkaBridge Farming

Security Assessment

March 24th, 2021

For :
PolkaBridge

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	PolkaBridge Farming
Description	DeFi
Platform	Ethereum; Solidity
Codebase	GitHub Repository
Commit	3b65dec59d9d5a5acd0ab5e0761791d0999e4acd 73f0e9b56c18ee407ad21b041a5ceda138c9212b 03c4a58d8dc04d0501f0f4ff349184c624fb6381

Audit Summary

Delivery Date	March. 22th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	March. 16th, 2021 - March. 22, 2021

Vulnerability Summary

Total Issues	10
Total Critical	0
Total Major	4
Total Minor	0
Total Informational	6
Notes	

Executive Summary

This report has been prepared for **PolkaBridge Farming** smart contract 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.

All of the functions in the protocol have proper access restriction and parameter sanitization where necessary. The equity was found to be calculated correctly for each of the accounts. Most of the findings are optimizational.

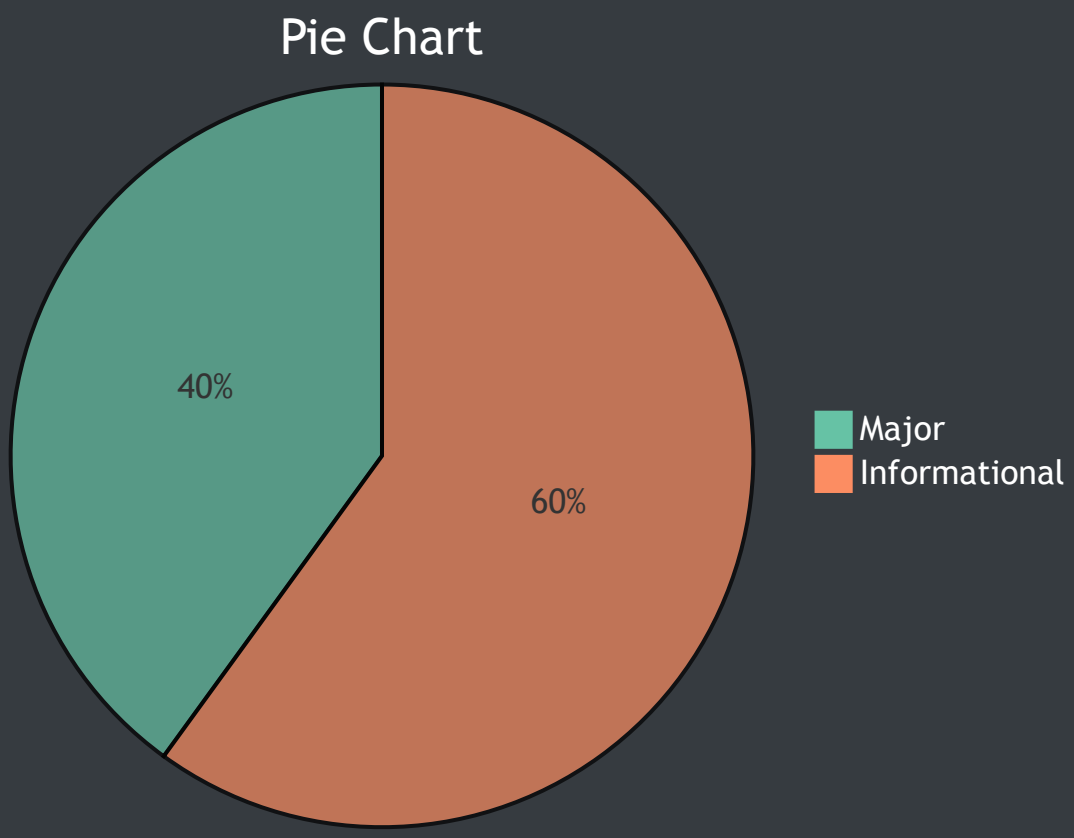
Additionally, to bridge the trust gap between administrator and users, administrator needs to express a sincere attitude with the consideration of the administrator team's anonymousness. The administrator has the responsibility to notify users with the following privileges of the `owner` :

- `owner` can add new pool with the arbitary values of parameter `_lpToken` , `_multiplier` and `_startBlock` through `add()` function in `PolkaBridgeMasterFarm.sol` smart contract.
- `owner` can update `multiplier` of any pool through `changeMultiplier()` function in `absorber.sol` smart contract.
- `owner` can active and stop any pool address through `activePool()` function and `stopPool()` function in `PolkaBridgeMasterFarm.sol` .

Client should inform any sensitive changes of project to project's community to improve the trustworthiness of the project. Moreover, any dynamic runtime changes on the protocol should be notified to the community. We also advise client to adopt Multisig, Timelock and/or DAO in the project.

File in Scope

ID	Contract	SHA-256 Checksum
PBF	PolkaBridgeMasterFarm.sol	e366758e4f2a803d5969106e8d4d916179e37406610c7aae98f64bd7cae603e5


Pie Chart
40%
60%
Major
Informational

ID	Title	Type	Severity	Resolved
PBF-01	Immutable State Variable	Coding Style	Informational	
PBF-02	`add()` Function Not Restricted	Volatile Code	Major	
PBF-03	Simplify Code	Optimization	Informational	
PBF-04	Simplify Code	Optimization	Informational	
PBF-05	Centralized Risk	Optimization	Major	
PBF-06	Missing Emit Events	Optimization	Informational	
PBF-07	SafeMath Not Used	Mathematical Operations	Informational	
PBF-08	Potential Reentrancy Vulnerability	Logic Issue	Major	
PBF-09	Proper Usage of public and external Type	Logic Issue	Informational	
PBF-010	Potential Reentrancy Vulnerability in `_harvest`	Logic Issue	Major	

PBF-01: Immutable State Variable

Type	Severity	Location
Coding Style	Informational	PolkaBridgeMasterFarm.sol: L39, L40

Description:

The state variable `START_BLOCK` only set one in the constructor should use immutable keyword

Recommendation:

We recommend using immutable for START_BLOCK in the declaration

```
1  uint256 public immutable START_BLOCK;
```

Alleviation:

N/A

PBF-02: add() Function Not Restricted

Type	Severity	Location
Volatile Code	Major	PolkaBridgeMasterFarm.sol: L58, L59

Description:

The comment in line L57, mentioned // XXX DO NOT add the same LP token more than once. Rewards will be messed up if you do.

The total amount of reward in function `updatePool()` will be incorrectly calculated if the same LP token is added into the pool more than once in function `add()`.

Recommendation:

Detect whether the given pool for addition is a duplicate of an existing pool. The pool addition is only successful when there is no duplicate. Using mapping of `addresses -> booleans`, which can restrict the same address being added twice.

Alleviation:

[PolkaBridge Team]: The team is addressed the issue in the commit 73f0e9b56c18ee407ad21b041a5ceda138c9212b

PBF-03: Simplify Code

Type	Severity	Location
Optimization	Informational	PolkaBridgeMasterFarm.sol: L88: L103

Description:

The function `getChangePoolReward` can simplify the code and logic.

Recommendation:

We recommend the function `getChangePoolReward` simplify as following:

```
function getChangePoolReward(uint256 _pid, uint256 _totalMultiplier) public view returns (uint256) {
    uint256 changePoolReward;
    if (_totalMultiplier == 0) {
        return 0;
    }

    uint256 currentPoolBalance = poolBalance();
    uint256 totalLastPoolReward = getTotalLastPoolReward();
    changePoolReward = ((currentPoolBalance.sub(totalLastPoolReward)).mul(poolInfo[_pid].multiplier).mul(1e18)).div(_totalMultiplier);


    return changePoolReward;
}

```

Alleviation:

[PolkaBridge Team]: The team is addressed the issue in the commit 73f0e9b56c18ee407ad21b041a5ceda138c9212b

PBF-04: Simplify Code

Type	Severity	Location
Optimization	Informational	PolkaBridgeMasterFarm.sol:L148: L152

Description:

The function `updatePool` can simplify the code and logic. The else branch in L150 is never be called from both of then internal and external of the contract.

Recommendation:

We recommend the if-else statement f L148 to L152 in function `updatePool` simplify as following:

```
if (flag == 1) {
    pool.lastPoolReward += _changePoolReward;
}
```

Alleviation:

[PolkaBridge Team]: The team is addressed the issue in the commit 73f0e9b56c18ee407ad21b041a5ceda138c9212b

PBF-05: Centralized Risk

Type	Severity	Location
Optimization	Major	PolkaBridgeMasterFarm.sol

Description:

`owner` is an important role in the contract. The owner address can operate on following functions:

- `add()`
- `stopPool()`
- `activePool()`
- `changeMultiplier`

Recommendation:

We advise the client to carefully manage project's private key and avoid any potential risks of being hacked. We also advise the client to adopt Multisig, Timelock and/or DAO in the project to manage sensitive role accesses.

Alleviation:

N/A

PBF-06: Missing Emit Events

Type	Severity	Location
Optimization	Informational	PolkaBridgeMasterFarm.sol

Description:

Function that affect the status of sensitive variables should be able to emit events as notifications to customers

- `claimReward()`
- `deposit()`
- `withdraw()`
- `emergencyWithdraw()`
- `stopPool()`
- `activePool()`
- `changeMultiplier()`

Recommendation:

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

```
function claimReward(uint256 _pid) public {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];
        ....
    emit ClaimRewardEvent(.....);
    user.rewardDebt = user.amountLP.mul(pool.accPBRPerShare).div(1e18);
}
```

Alleviation:

[PolkaBridge Team]:

- `claimReward()` is missing the emit event
- The team is addressed the rethe issue in the commit 73f0e9b56c18ee407ad21b041a5ceda138c9212b

PBF-07: SafeMath Not Used

Type	Severity	Location
Mathematical Operations	Informational	PolkaBridgeMasterFarm.sol

Description:

`SafeMath` from OpenZeppelin is not used in the following functions which makes them possible for overflow/underflow and will lead to an inaccurate calculation result.

- `updatePool()`
- `_harvest()`
- `add()`
- `getTotalLastPoolReward()`
- `countTotalMultiplier()`
- `avgRewardPerBlock()`

Recommendation:

Considering use OpenZeppelin's SafeMath library for all of the `uint` operations.

Reference:
https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/math/SafeMath.sol

Alleviation:

[PolkaBridge Team]: The team is addressed the issue in the commit 73f0e9b56c18ee407ad21b041a5ceda138c9212b

PBF-08: Potential Reentrancy Vulnerability

Type	Severity	Location
Logic Issue	Major	PolkaBridgeMasterFarm.sol L313, L319, L325

Description:

The function is exposed to reentrancy attack where attacker is able to reentrant functions multiple times in single transaction.

- `claimReward()`
- `deposit()`
- `withdraw()`

Recommendation:

We advise the client to consider to adopt following `nonReentrant` modifier on above mentioned functions:

Reference: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/security/ReentrancyGuard.sol

Alleviation:

[PolkaBridge Team]: The team is addressed the issue in the commit 73f0e9b56c18ee407ad21b041a5ceda138c9212b

PBF-09: Proper Usage of `public` and `external` Type

Type	Severity	Location
Logic Issue	Informational	PolkaBridgeMasterFarm.sol L313, L319, L325

Description:

Following public functions that are never called by the contract could be declared external . When the inputs are arrays external functions are more efficient than public functions. When the inputs are arrays external functions are more efficient than public functions.

- `claimReward()`
- `deposit()`
- `withdraw()`
- `emergencyWithdraw()`
- `getPoolInfo()`
- `stopPool()`
- `activePool()`
- `changeMultiplier()`
- `countActivePool()`
- `countTotalMultiplier()`
- `totalRewardClaimed()`
- `avgRewardPerBlock()`

Recommendation:

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

Alleviation:

N/A

PBF-10: Potential Reentrancy Vulnerability in `_harvest`

Type	Severity	Location
Logical Issue	Major	PolkaBridgeMasterFarm.sol L206: L208

Description:

The function is exposed to reentrancy attack where attacker is able to reentrant functions multiple times in single transaction.

- `_harvest()`

Recommendation:

We advise client to consider to adopt following `nonReentrant` modifier on above mentioned functions:

Reference: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/security/ReentrancyGuard.sol

or changing the order of handling the `transfer` function call:

```
function _harvest(uint256 _pid) internal {
    ...
    if (pending > 0) {
        pool.lastPoolReward = pool.lastPoolReward.sub(pending);
        pool.totalRewardClaimed = pool.totalRewardClaimed.add(pending);
        polkaBridge.transfer(msg.sender, pending);
    }
    ...
}
```

Alleviation:

[PolkaBridge Team]: The team is addressed the issue in the commit 03c4a58d8dc04d0501f0f4ff349184c624fb6381

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.