CERTIK

Cub Finance

Security Assessment

Mar 30th 2021

Summary

This report has been prepared for Cub Finance smart contracts, MasterChef, CubToken, Timelock and libs to discover issues and vulnerabilities in the source code as well as any dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing 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 security experts.

The security assessment resulted in 7 findings that ranged from minor to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices.

We suggest below recommendations that could better serve the project from the security perspective:

1. Enhance general coding practices for better structures of source codes;
2. Add enough unit tests to cover the possible use cases given they are currently missing in the repository;
3. Provide more comments per each function for readability, especially contracts are verified in public;
4. Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Name	Cub Finance
Codebase	https://github.com/CubFinance/contracts/
Commit Hash	e9df9a03001d880c3e512387ad987f6b7bc6113b

Engagement Summary

Delivery Date	Mar 30th, 2021
Methodology	Static analysis and manual review
Contracts in Scope	3
Contract - Token	CubToken
Contract - MasterChef	MasterChef
Contract - TimeLock	TimeLock

Finding Summary

Total	7
Critical	1
Medium	0
Minor	2
Informational	4

Findings

ID	Title	Severity	Response
CTK-CUB-1	Checks Effects Interaction Pattern Not Used	Minor	Resolved
CTK-CUB-2	Function Return Value Ignored	Informational	Resolved
CTK-CUB-3	Missing Emit Events	Informational	Resolved
CTK-CUB-4	add() Function Not Restricted	Critical	Acknowledged
CTK-CUB-5	Lack of Input Validation	Minor	Resolved
CTK-CUB-6	Privileged Ownerships on MasterChef	Informational	Resolved
CTK-CUB-7	Privileged Ownerships on CubToken	Informational	Resolved

CTK-CUB-1 | Checks Effects Interaction Pattern Not Used

Type	Severity	Location
Logic Issue	Minor	MasterChef: L181

Description

In function `add()`, `lpToken` is pointing to a smart contract that is implemented based on a BEP20 interface. This smart contract can only be passed into function `add()` by `owner` as one of the parameters while the implementation of `lpToken` is unknown statically, even if `lpToken` strictly followed the BEP20 interface.

Due to the unknown implementation of contract `lpToken`, the implementation of function `safeTransfer()` is also unknown and may have a malicious logical implementation that calls back to the function `deposit()`, which can lead to another invocation of `safeTransfer()` in L181 without updating `user.amount` in L182. This is dangerous to the `user.amount` and will incorrectly calculate the user's balance eventually.

Recommendation

We advise developers to swap `pool.lpToken.safeTransfer(feeAddress, depositFee);` and `user.amount = user.amount.add(_amount).sub(depositFee);` to follow the Checks-Effects-Interactions Pattern.

```
function deposit(uint256 _pid, uint256 _amount) public {
  ...
  if(_amount > 0) {
    pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount);
    if(pool.depositFeeBP > 0){
      uint256 depositFee = _amount.mul(pool.depositFeeBP).div(10000);
      user.amount = user.amount.add(_amount).sub(depositFee);
      pool.lpToken.safeTransfer(feeAddress, depositFee);
    }else{
      user.amount = user.amount.add(_amount);
    }
  }
  ...
}
```

Alleviation

The update is applied at a later commit.

CTK-CUB-2 | Function Return Value Ignored

Type	Severity	Location
Volatile Code	Informational	MasterChef: L224, 226

Description

1. Return values of function `transfer()` are ignored in function `safeCubTransfer()`.
2. The return values of `cub.transfer(_to, cubBal),cub.transfer(_to, _amount);` are ignored in function `safeEggTransfer()`.

Recommendation

1. We advise developers to handle the return value of `transfer()` to check if the transfer is executed without any error.
2. Using `cub.safeTransfer()` Instead of `cub.transfer()`.

Alleviation

The update is applied at a later commit.

CTK-CUB-3 | Missing Emit Events

Type	Severity	Location
Volatile Code	Informational	MasterChef L231, L236. L242

Description

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

MasterChef:

`dev()`, `setFeeAddress()`, `updateEmissionRate()`

Recommendation

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

```
event SetFeeAddress(address indexed user, address indexed _feeAddress);
...
function setFeeAddress(address _feeAddress) public{
  require(msg.sender == feeAddress, "setFeeAddress: FORBIDDEN");
  feeAddress = _feeAddress;
  emit SetFeeAddress(msg.sender, _feeAddress)
}
```

Alleviation

The update is applied at a later commit.

CTK-CUB-4 | add() Function Not Restricted

Type	Severity	Location
Volatile Code	Critical	MasterChef: L93

Description

The comment in L92, 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 `cubReward` in function `updatePool()` will be incorrectly calculated if the same LP token is added into the pool more than once in function `add()`.

However, the code is not reflected in the comment behaviors as there isn't any valid restriction on preventing this issue.

The current implementation is relying on the trust of the owner to avoid repeatedly adding the same LP token to the pool, as the function will only be called by the owner.

Recommendation

Using mapping of `addresses -> booleans`, which can restrict the same address being added twice.

Alleviation

The update is applied at a later commit.

CTK-CUB-5 | Missing Zero Address Validation

Type	Severity	Location
Volatile Code	Minor	MasterChef: L231, L236

Description

The assigned value to `devaddr`, `feeAddress` should be verified as non zero value to prevent being mistakenly assigned as `address(0)` in `dev()` function and `setFeeAddress()`. Violation of this may cause losing ownership of `devaddr`, `feeAddress`.

Recommendation

Check that the address is not zero by adding checks in function `dev()` and `setFeeAddress()`. Please ignore if the team inclines to leverage the same function in a way to renounce the fee collections (mimic the token burn in a way).

Alleviation

The update is applied at a later commit.

CTK-CUB-6 | Privileged Ownerships on MasterChef

Type	Severity	Location
Business Model	Informational	MasterChef: L93, L110, L242

Description

The owner of MasterChef has permission to add and set pools that could update the parameters on rewards without obtaining the consensus of the community.

Recommendation

Renounce ownership when it is the right timing, or gradually migrate to a timelock plus multisig governing procedure and let the community monitor in respect of transparency considerations.

Alleviation

The team confirms that the owner of masterchef is a timelock contract.

CTK-CUB-7 | Privileged Ownerships on CubToken

Type	Severity	Location
Business Model	Informational	CubToken

Description

CubToken is the standard BEP20 implementation that contains the mint functionality with ownership controls, which means whoever obtained access to the owner account would be able to tamper with the integrity of the token economics.

Recommendation

In general, renounce ownership when it is the right timing, or gradually migrate to a timelock plus multisig governing procedure and let the community monitor in respect of transparency considerations. Specifically for this scenario, we assume the owner will be transferred to the vault (MasterChef) on top of the token. We recommend that the team maintains a high level of transparency on such a transaction taking place.

Alleviation

The team confirmed that the token owner is the MasterChef.

Appendix | Finding Categories

Gas Optimization

Refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction in the total gas cost of a transaction.

Mathematical Operations

Refer to exhibits that relate to mishandling of math formulas, such as overflows, incorrect operations, etc.

Logical Issue

Refer to exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how `block.timestamp` works.

Control Flow

Concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

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 in-storage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete`.

Coding Style

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

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

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

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.

Business Model

Refer to contract or function logics that are debatable or not clearly implemented according to the design intentions.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK's prior written consent.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is 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 assessment. This report does 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.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing 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.

About CertiK

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.

CERTIK

Provable Trust For All