CRYPTOBIRDS
PLATFORM

# Smart contract security audit

# CryptoBirds

v.1.2


No part of this publication, in whole or in part, may be reproduced, copied, transferred or any other right reserved to its copyright a CTDSec, including photocopying and all other copying, any transfer or transmission using any network or other means of communication, in any form or by any means such as any information storage, transmission or retrieval system, without prior written permission.

# Table of Contents



Copyright © CTDSec - All rights reserved

# 1.0 Introduction

## 1.1 Project engagement

During June of 2021, CryptoBirds engaged CTDSec to audit smart contracts that they created. The engagement was technical in nature and focused on identifying security flaws in the design and implementation of the contracts. CryptoBirds provided CTDSec with access to their code repository and whitepaper.

## 1.2 Disclaimer

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that's able to withstand the network's fast-paced and rapidly changing environment, we at CTDSec recommend that CryptoBirds team put in place a bug bounty program to encourage further and active analysis of the smart contract.

Copyright © CTDSec - All rights reserved

# 2.0 Coverage

## 2.1 Target Code and Revision

For this audit, we performed research, investigation, and review of the CryptoBirds contract followed by issue reporting, along with mitigation and remediation instructions outlined in this report. The following code files are considered in-scope for the review:

Source:

Package 03_05_2021_CryptoBirds.zip [SHA256]:

666439b28bde214f35ec069b3d55099421efd8ddb9b68eccf40324f3cb75d8e0

- IStakingRewards.sol
- LPToken.sol
- StakingRewards.sol
- StakingRewardsFactory.sol
- IVesting.sol
- IXCB.sol
- Vesting.sol
- XCB.sol
- Platform.sol

Copyright © CTDSec - All rights reserved

## 2.2 Attacks made to the contract

In order to check for the security of the contract, we tested several attacks in order to make sure that the contract is secure and follows best practices.

| № | Issue description. | Checking status |
|---|---|---|
| 1 | Compiler warnings. | PASSED |
| 2 | Race conditions and Reentrancy. Cross-function race conditions. | PASSED |
| 3 | Possible delays in data delivery. | PASSED |
| 4 | Oracle calls. | PASSED |
| 5 | Front running. | PASSED |
| 6 | Timestamp dependence. | PASSED |
| 7 | Integer Overflow and Underflow. | PASSED |
| 8 | DoS with Revert. | PASSED |
| 9 | DoS with block gas limit. | PASSED |
| 10 | Methods execution permissions. | PASSED |
| 11 | Economy model. If application logic is based on an incorrect economic model, the application would not function correctly and participants would incur financial losses. This type of issue is most often found in bonus rewards systems, Staking and Farming contracts, Vault and Vesting contracts, etc. | SOLVED BY DEV TEAM |
| 12 | The impact of the exchange rate on the logic. | PASSED |
| 13 | Private user data leaks. | PASSED |
| 14 | Malicious Event log. | PASSED |
| 15 | Scoping and Declarations. | PASSED |
| 16 | Uninitialized storage pointers. | PASSED |
| 17 | Arithmetic accuracy. | PASSED |

Copyright © CTDSec - All rights reserved

| | | |
|---|---|---|
| 18 | Design Logic. | PASSED |
| 19 | Cross-function race conditions. | PASSED |
| 20 | Safe Zeppelin module. | PASSED |
| 21 | Fallback function security. | PASSED |
| 22 | Overpowered functions / Owner privileges | PASSED |

Copyright © CTDSec - All rights reserved

# 3.0 Security Issues

## 3.1 High severity issues [0]

**1. Time comparison**

Issue:

The functions claimReward and getPendingReward compare invested time plus 90 days with current time in a wrong way.

```
ftrace | funcSig
function claimReward() public {
  Payment[] storage myPayments = paymentsHistory[msg.sender];
  require(myPayments.length > 0, "No pending payments");

  uint256 amountToTransfer = 0;
  for (uint256 i = 0; i < myPayments.length; i++) {
    if(myPayments[i].timestamp + 90 days > block.timestamp){
      amountToTransfer = amountToTransfer.add(myPayments[i].amount);
    }
    tokensLocked = tokensLocked.sub(myPayments[i].amount);
  }
```

```
  function getPendingReward() public view returns (uint256) {
    Payment[] memory myPayments = paymentsHistory[msg.sender];
    require(myPayments.length > 0, "No pending payments");
    uint256 amountToTransfer = 0;
    for (uint256 i = 0; i < myPayments.length; i++) {
      //IF THE PAYMENT IS OLDER THAN 3 MONTHS
      if(myPayments[i].timestamp + 90 days > block.timestamp){
        amountToTransfer = amountToTransfer.add(myPayments[i].amount);
      }
    }
    return amountToTransfer;
  }
```

Dev team update: The development team has improved the documentation to avoid misunderstandings that the function is executing correctly.

Copyright © CTDSec - All rights reserved

## 3.2 Medium severity issues [0]

No medium severity issues found.

## 3.3 Low severity issues [1]

**1.Out of gas**

Issue:

The functions claimReward and getPendingReward use the loop to collect the total amount to transfer. Functions will be aborted with OUT_OF_GAS exception if there will be a long payments list.

Recommendation:

Check that the payment array length is not too big.

## 3.3 Informational [1]

***Constant variable***

In XCB.sol file minted variable is constantly equals to false.

We recommend removing that variable and dependent require function.

Copyright © CTDSec - All rights reserved

```
bool private minted = false;

ftrace | funcSig
constructor(string memory name_, string memory symbol_) ERC20(name_, symbol_){}

/**
 * @dev Set the vesting contract to assign the total supply
 * @param _vestingContract the receiver address of the vesting contract
 * @param _totalSupply the amount of total token to vest
 */
ftrace | funcSig
function setVestingContract(address _vestingContract, uint _totalSupply) public onlyOwner{
    require(!minted, "Already minted");
    _mint(_vestingContract, _totalSupply);
}
```

# 4.0 Owner Privileges

- Owner can start the vesting process.
- Owner or Team Member can release platform vesting, make distribution for several addresses, pull or burn unclaimed tokens.

Copyright © CTDSec - All rights reserved

# 5.0 Summary of the audit

Smart contracts contain low severity issues and it’s safe to deploy.

Copyright © CTDSec - All rights reserved