THE ARCADIA GROUP

Audit of the Cover Protocol Contracts

a report of findings by

Van Cam Pham, PhD

innovative fortuna iuvat

December 1st, 2020

Table of Contents

Document Info

Client	Cover Protocol
Title	Smart Contract Audit of Cover Protocol Contracts
Auditors	Van Cam Pham, PhD
Reviewed By	Joel Farris
Approved By	Rasikh Morani

Contact

For more information on this report, contact The Arcadia Media Group Inc.

Rasikh Morani
(972) 543-3886
rasikh@arcadiamgroup.com
https://t.me/thearcadiagroup

Executive Summary

A Representative Party of the Cover Protocol ("Cover Protocol") engaged The Arcadia Group ("Arcadia"), a software development, research, and security company, to conduct a review of the following Cover Protocol smart contracts on the Cover Protocol repo at Commit #c31e93c32b2b2035fa73cf3a51842cfe64d76e99.

BlackSmith.sol
COVER.sol
Migrator.sol
Vesting.sol
MerkleProof.sol

After a first review and report and discuss our findings with the team. The team fixed all issues found by our findings and we had a second view on the code at commit 0f5deb2083b47ad259668968e203399616c2bd78.

Arcadia completed the review using various methods primarily consisting of dynamic and static analysis. This process included a line by line analysis of the in-scope contracts, optimization analysis, analysis of key functionalities and limiters, and reference against intended functionality.

Conclusion

Based upon the findings and analysis, Cover has followed our recommendations and addressed the issues reflected in this document in the commit hash provided above and the hashes provided in the issues defined below.

Findings

1. Function `collectDust` can claim bonus token amount to treasury any time

- COVER-1
- Severity: Medium
- Likelihood: Medium
- Impact: Medium

- Target: BlackSmith.sol
- Category: Medium
- Finding Type: Dynamic
- Lines: 236-250

In the BlackSmith contract, the function `collectDust` is assumed to transfer all funds, which are not bonus tokens, to the treasury. The function can be called by any one and the dust funds will be sent to the treasury address upon calling the function.

However, as the contract owner can modify `allowBonusTokens` for any token, which bypasses the second `require` check in the function. By modifying allowBonusTokens, anyone can call `collectDust` at any time, and all bonus funds reversed for shield mining will be sent to the treasury.

```
function collectDust(address _token) external override {
   Pool memory pool = pools[_token];
   require(pool.lastUpdatedAt == 0, "Blacksmith: lpToken, not allowed");
   require(allowBonusTokens[_token] != 1, "Blacksmith: active bonusToken, not allowed");

   IERC20 token = IERC20(_token);
   uint256 amount = token.balanceOf(address(this));
   require(amount > 0, "Blacksmith: 0 to collect");

   if (_token == address(0)) { // token address(0) == ETH
     payable(treasury).transfer(amount);
   } else {
     token.safeTransfer(treasury, amount);
   }
}
```

Action Recommended:

- Collecting dust for bonus tokens should only be allowed when the end time for bonus token shield mining completes. Thus the function should ensure that bonus dust tokens should only be withdrawn after the end time for shield mining of the token.
- The function should only transfer bonus tokens to the treasury if users already withdraw their bonus.

Review of the code at commit 0f5deb2083b47ad259668968e203399616c2bd78:

- The original collectDust function is now divided into two functions:
  - One is for collecting non-bonus dust token
  - One is for collecting bonus dust token
- Function collectBonusDust in this commit only allows for withdrawing dust bonus token to the treasure only 1 week after the completion of shield mining for the bonus token. The one-week period is to allow users to claim rewards.

```
function collectDust(address _token) external override {
   Pool memory pool = pools[_token];
   require(pool.lastUpdatedAt == 0, "Blacksmith: lpToken, not allowed");
   require(allowBonusTokens[_token] == 0, "Blacksmith: bonusToken, not allowed");

   IERC20 token = IERC20(_token);
   uint256 amount = token.balanceOf(address(this));
   require(amount > 0, "Blacksmith: 0 to collect");

   if (_token == address(0)) { // token address(0) == ETH
     payable(treasury).transfer(amount);
   } else {
     token.safeTransfer(treasury, amount);
   }
 }

 /// @notice collect bonus token dust to treasury
 function collectBonusDust(address _lpToken) external override {
   BonusToken memory bonusToken = bonusTokens[_lpToken];
   require(bonusToken.endTime.add(WEEK) < block.timestamp, "Blacksmith: bonusToken, not
ready");

   IERC20 token = IERC20(bonusToken.addr);
   uint256 amount = token.balanceOf(address(this));
   require(amount > 0, "Blacksmith: 0 to collect");
   token.safeTransfer(treasury, amount);
}
```

2. Function `addBonusToken` can wipe out all existing unclaimed bonus

- COVER-2
- Severity: High
- Impact: High

- Target: BlackSmith.sol
- Category: Informational
- Finding Type: Static
- Lines: 204-233

Functions `addBonusToken` can be called by anyone in order to add bonus token funds for mining. The function is expected to add the specified amount of the specified bonus token only if the last mining period for the same liquidity (LP) token finishes.

However, there is a bug in function `addBonusToken` that could wipe out all unclaimed bonus funds of the current bonus token. Specifically, the statement at line 218 `require(bonusToken.balanceOf(address(this)) == 0, "Blacksmith: last bonus not all claimed");` should check for the current token balance of the contract, the new bonus token.

```
    function addBonusToken(
    address _lpToken,
    address _bonusToken,
    uint256 _startTime,
    uint256 _endTime,
    uint256 _totalBonus
  ) external override {
    IERC20 bonusToken = IERC20(_bonusToken);
    require(pools[_lpToken].lastUpdatedAt != 0, "Blacksmith: pool does NOT exist");
    require(allowBonusTokens[_bonusToken] == 1, "Blacksmith: bonusToken not allowed");

    BonusToken memory currentBonusToken = bonusTokens[_lpToken];
    if (currentBonusToken.totalBonus != 0) {
      require(currentBonusToken.endTime < block.timestamp, "Blacksmith: last bonus period
hasn't ended");
      require(bonusToken.balanceOf(address(this)) == 0, "Blacksmith: last bonus not all
claimed");
    }
```

```
   require(_startTime >= block.timestamp && _endTime > _startTime, "Blacksmith: messed up
timeline");
   require(_totalBonus > 0 && bonusToken.balanceOf(msg.sender) >= _totalBonus, "Blacksmith:
incorrect total rewards");

   bonusTokens[_lpToken] = BonusToken({
     addr: _bonusToken,
     startTime: _startTime,
     endTime: _endTime,
     totalBonus: _totalBonus,
     accBonusPerToken: 0,
     lastUpdatedAt: _startTime
   });
   bonusToken.safeTransferFrom(msg.sender, address(this), _totalBonus);
 }
```

Action Recommended: The bug can be fixed checking the contract balance for the current token.

Review of the code at commit 0f5deb2083b47ad259668968e203399616c2bd78:

- The bug was fixed in the updated commit:

```
BonusToken memory currentBonusToken = bonusTokens[_lpToken];
   if (currentBonusToken.totalBonus != 0) {
     require(currentBonusToken.endTime.add(WEEK) < block.timestamp, "Blacksmith: last bonus
period hasn't ended");
     require(IERC20(currentBonusToken.addr).balanceOf(address(this)) == 0, "Blacksmith: last
bonus not all claimed");
   }
```

3. Function `addBonusToken` fails even if all users already claim bonus token

- COVER-3
- Severity: Low
- Impact: Low
- Target: BlackSmith.sol
- Category: Informational
- Finding Type: Static
- Lines: 304-233

Function addBonusToken can fail even if all users already claim rewards. This is due to approximation for math operands like division, which cause the current bonus token balance of the contract to not be 0 precisely. This causes the `require` statement in 218 fails even if all users already make bonus token claims.

```
    require(bonusToken.balanceOf(address(this)) == 0, "Blacksmith: last bonus not all
claimed");
```

Action Recommended: The issue was discussed with the development team of Cover Protocol. The issue was mitigated by having a `collectBonusDust` function that transfers all bonus dusts to the treasury. This function effectively makes the bonus token balance of the contract become 0.

4. Function `addBonusToken` should check for deflationary tokens

- COVER-5
- Severity: Medium
- Likelihood: Medium
- Impact: Medium
- Target: BlackSmith.sol
- Category: Informational
- Finding Type: Dynamic
- Lines: 224-232

Deflationary tokens have fees or burn on transfer, function `addBonusToken` should check to ensure that the received bonus token amount must not exceed the bonus added to the pool in the calculation.

Specifically, at line 232, the contract might receive less than _totalBonus due to the deflation of the bonus token.

```
bonusTokens[_lpToken] = BonusToken({
     addr: _bonusToken,
     startTime: _startTime,
     endTime: _endTime,
     totalBonus: _totalBonus,
     accBonusPerToken: 0,
     lastUpdatedAt: _startTime
   });
bonusToken.safeTransferFrom(msg.sender, address(this), _totalBonus);
```

Action Recommended: It is safe to check that the expected total bonus added to the pool must be less equal to the actual bonus token amount the pool receives by transfer.

Review of the code at commit 0f5deb2083b47ad259668968e203399616c2bd78:

- Safe validation for the amount of actual receive token is added to the code.

```
uint256 balanceBefore = bonusToken.balanceOf(address(this));
   bonusToken.safeTransferFrom(msg.sender, address(this), _totalBonus);
   uint256 balanceAfter = bonusToken.balanceOf(address(this));
   require(balanceAfter > balanceBefore, "Blacksmith: incorrect total rewards");

   bonusTokens[_lpToken] = BonusToken({
     addr: _bonusToken,
     startTime: _startTime,
     endTime: _endTime,
     totalBonus: balanceAfter.sub(balanceBefore),
     accBonusPerToken: 0,
     lastUpdatedAt: _startTime
   });
```

Disclaimer

While best efforts and precautions have been taken in the preparation of this document, The Arcadia Group and the Authors assume no responsibility for errors, omissions, or damages resulting from the use of the provided information. Additionally, Arcadia would like to emphasize that the use of Arcadia's services does not guarantee the security of a smart contract or set of smart contracts and does not guarantee against attacks. One audit on its own is not enough for a project to be considered secure; that categorization can only be earned through extensive peer review and battle testing over an extended period.