SOMISH

Blockchain Labs

Smart Contract Audit Report

PLincHub IPO Platform

June 1st, 2019

Index

Summary

Audit Report prepared by Somish Blockchain Labs for PLincHub covering the IPO smart contract.

Process and Delivery

Two (2) independent experts performed an unbiased and isolated audit of the code below. The debrief took place on June 1st, 2019, and the final results are presented here.

Audited Files

The following contract(s) were covered during the audit:

- IPO.sol

Client Documents

The following document(s) were provided by the client for the purpose of the audit:

- Main - PLincHub Red Paper available at the following link: https://hub.plinc.io/assets/redpaper/PLincHubRedPaper.pdf
- Supporting - Article on Steemit at the following link: https://steemit.com/crowdfunding/@spielley/crowdfunding-reimagined

Notes

Audit was performed on contract address **0x8d943e036c1074d464e5510823f0f6ab96f969d6** verified on **Rinkeby network** and pragma version 0.4.25

Intended Behavior

Profit line inc. Hub (PLinc Hub) is an intense dividend dapp using the same innovative bonds mechanics from profit line inc. at plinc.io. Unlike standard PLinc PLinc Hub returns divs very quickly using a 98% redistribution rate of new ETH into the game. The goal is to achieve an equal redistribution of ETH across all bonds holders where they can get in anytime and are always early to the party. PLinc Hub is the prime hub where later dapps and Initial PLinc offerings (IPOs) will plug in - to gain the dividend benefit from investors who come afterward. As long as the mechanisms within the PLinc Hub machine are turning from player activity, they'll accrue passive income and generate up to a 10% profit from their initial investment. In order to recycle and compound profits, users can reinvest funds.

The system revolves around the following concept:

1. Bonds

   Bonds are the fuel that powers the PLinc machine, users need to buy these in order to participate in PLinc Hub. Players buy bonds with ETH and these bonds will begin to accrue ETH as more ETH enters the game. Players can then trade back fully or partially filled bonds for ETH through their player vault.

2. The player vault

   The player vault holds earned ETH which you can withdraw directly to your ethereum wallet. One needs to transfer filled bonds to player vault to allow withdrawal of earned ETH to the wallet. The ETH balance available to transfer to the player vault increases as your bonds are filled.

3. The piggy bank

   The piggy holds ETH that is profited through reinvesting. When a user sets a % amount to reinvest that amount is used to purchase new bonds. The actual profit is sent to a piggy bank where it can be withdrawn directly to ETH wallet.

4. The exit pot

   If for any reason a player needs to withdraw ETH immediately before their bonds have been filled they can do so with the 'Exit PLinc with loss' function. When utilized this function will allow to reduce part or all of bonds for a loss.

5. Playing style

   PLinc players have a range of approaches they can adopt in order to profit. Below are the 3 distinct ways in which you can play. Players can switch between all 3 methods whenever they like: Hodler Players simply purchase bonds and wait for them to be filled as new players enter the game. Hodlers can also activate the auto reinvest function which will allow freelancers to recycle their profits buying back more bonds thus compounding their profit. See the Auto-reinvest page for more details Investor Players who choose to buy a management position to gain additional payouts as new players enter the game and players who invest in the SPASM smart contract. See the Investment opportunities page for more details Freelancer Any player can choose to undertake freelance tasks for PLinc Hub. Tasks undertaken by freelancers help to keep the PLinc Hub machine running, delivering the network transactions required to distribute ETH throughout the game. Completing freelance tasks benefits PLinc Hub players in 2 ways: 1. It burns the total supply of other players bonds, increasing your % share. This helps your unfilled bonds to fill with dividends more quickly. 2. You get paid 0.1% in unfilled bonds for every fill you complete.

6. Investment opportunities

Management positions

For players who wish to earn via an alternative method to purchasing bonds, PLinc Hub offers management positions. Management positions get dividends from PLinc's internal fund distribution. The management table on the dashboard specifies what % of dividends player receives from the management fund distribution. The player can buy a management position from another player 110% of the price the current owner paid.

SPASM

PLinc is part of Spielleys Profit Allocation Share Module(SPASM), the dev fee taken within PLinc, is shared across all SPASM token holders (shareholders of PLinc). SPASM is a crowdfunding token to try and make Spielley a fulltime dev. Eth dapps that are made by Spielley will incorporate SPASM for the dev fee payout. Players can buy spasm here: https://oneclickdapp.com/spasmtoken/ Or check out the contract at: https://etherscan.io/address/0xfaae60f2ce6491886c9f7c9356bd92f688ca66a1#code

7. Freelancing

All PLinc players are invited to freelance on behalf of PLinc. Whilst offering player advantages in doing so it also keeps the PLinc machine running, distributing ETH to other players. There are 3 tasks available to freelancers:

a. Fetch: Moves all the players available divs to fills
b. Fill: Sends the players fills balance to their player vault whilst burning their bonds
c. Freelance: Send the players player vault balance to their piggy bank whilst minting them new bonds

Issues Found

Critical

1. Code Deadlock

The contract allows several players to purchase stakes via buyStake() method. However, the process does not move forward until the hubFundAdmin calls fetchHubVault() function. If somehow, hubFundAdmin doesn't perform his/her responsibilities, all the funds will remain stuck perpetually within the hub contract.

2. Reentrancy attack in buyStake

The function buyStake() transfers funds to a contract and calls its function buyBonds(). If the function buyBonds() calls the function buyStake() again, the system will undergo a reentrancy attack. For x amount of ETH, it will issue x bonds to the sender and n*x bonds to the contract, n being the number of times the code is reentered. Since the hub contract was not provided during this audit, the function buyStake() is vulnerable to such an attack. We suggest using a reentrancy check here.

3. Transfer of funds to unknown Non-verified contract

The function buyStake() transfers funds to another other contract i.e. the hub contract, which is not provided for this audit and is not verified on Rinkeby Etherscan. Since the intention of the hub contract is unknown, it leaves the IPO contract vulnerable.

4. Player can get profit > 10%

A player can execute fetchdivs() function any number of times without executing fillBonds() function. In such a scenario, bondsOutstanding will not reduce until bonds get filled. So the player can fetch more than 110% of his bonds which is not the intended behavior as per the Red Paper.

5. Not reducing stakes owned by the player after moving dividend stake to player vault

In fetchdivsStake ETH is transferred to playerVault but not reduced from accountsStake[account].owned. Hence, the player can execute fetchdivsStake() function any number of times with the same accountsStake[account].owned.

Major

1. Overflowing of Integers

The dividendsOwing() and dividendsOwingStake() functions do not use SafeMath for all operations, which can result in overflow of integer values. Enforce usage of SafeMath in every function.

2. No logic found for auto reinvest

According to the provided Red Paper, any player can apply for auto reinvest but the smart contract doesn't have any logic for the same.

3. No logic found for purchase of management seat

According to the provided Red Paper, any player can purchase a management seat but the smart contract doesn't have any logic for the same.

4. Denial of service

Frontrunning can happen if every time a player waits for the hubFundAdmin to fill the pot and fetches his bond immediately with a high gas price. In such a scenario, other players may end up busy-waiting.

Minor

1. The redistribution rate is 95%

According to the Red Paper, redistribution rate should be 98%. However in potToPayout() function investors receive only 95%.

2. No way to transfer ownership

While there is an admin which is responsible for performing several functions, there is no way to transfer his/her administrative rights. There should be some way to transfer ownership in case of some unforeseen problems with the present owner account.

Consider using Ownable.sol from OpenZeppelin.

3. Equality condition is missing

In potToPayout function there is no branch which handles the condition (value == totalSupplyBonds). It is a recommended practice to cover every possible branch.

Notes

1. Upgrade code to Solidity version > 0.5.7

Currently, the smart contract is written in Solidity version 0.4.25 which is now a known vulnerable version of Solidity. The contracts should be updated and their compiler version should be fixed.

Recommendation
Refer to the Ethereum changelog for the most up to date Solidity version. Contract elements may need to be updated in order to work with the latest Solidity version. Ethereum changelog can be accessed on the following link:

https://github.com/ethereum/solidity/blob/develop/Changelog.md

2. Code commenting

Consider adding comments to functions and variable declarations in the smart contracts - it gives clarity to the reader and reduces confusion for the developers as well.

3. Hard-coded constants and types

Consider using constants for constant values instead of hardcoding. For example 20,19 in potToPayout() function etc.

Recommendation
For readability purposes, the usage of a constant type is suggested.

4. Add reasons for revert

Consider adding messages for require or revert statements, it gives clarity to the reader.

5. Follow the Solidity style guide

Consider following the Solidity style guide. It is intended to provide coding conventions for writing solidity code.

Recommendation
You can find documentation for Solidity Style guide on the following link:
https://solidity.readthedocs.io/en/v0.4.24/style-guide.html

6. Violation of check-effect-interaction pattern

One of the best practices is to follow the check-effect-interaction pattern. It is recommended to avoid state changes after external calls. Refer to fetchHubVault() function.

7. Multiplication after division

Solidity operates only with integers. Thus, if the division is done before the multiplication, the rounding errors can increase dramatically. For example in the potToPayout() function.

Recommendation
It is a better practice to use multiplication before divide. It will reduce rounding off errors.

8. Gas Optimizations

Consider using keywords like **internal** and **external** instead of **public,** it will reduce gas consumption.

Recommendation
Consider using **internal** for the function which is callable only within the same contract and using **external** for the function which is callable only from outside of contract. It will optimize the function call as well as gas required for deployment.

Closing Summary

Upon audit of the PLincHub's IPO smart contract, it was observed that the contracts contain various critical, major and minor issues, along with several areas of notes.

We recommend that the critical, major and minor issues should be resolved. Resolving for the areas of notes are up to PLincHub's discretion. The notes refer to improving the operations of the smart contract.

Disclaimer

Somish Blockchain Labs's audit is not a security warranty, investment advice, or an endorsement of the PLincHub's IPO platform or its products. This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors of Somish from legal and financial liability.

Somish Solutions Limited