Audit of The Poolz Finance Contracts

a report of findings by

Van Cam Pham, PhD

innovative fortuna iuvat

December 17th, 2020

Table of Contents

Document Info

Client	Poolz Finance
Title	Smart Contract Audit of Poolz Finance Contracts
Auditors	Van Cam Pham, PhD
Approved By	Rasikh Morani

Contact

For more information on this report, contact us below

Rasikh Morani
rasikh@arcadiamgroup.com
https://t.me/thearcadiagroup

Executive Summary

A Representative Party of Poolz Finance ("Poolz") engaged The Arcadia Group ("Arcadia"), a software development, research, and security company, to conduct a review of the following Poolz Finance smart contracts on the Poolz repo at Commit #d78a261077b9c5495680b7fd80b73f3cc0f9a1c1.

ERC20Helper.sol
ETHHelper.sol
Invest.sol
InvestorData.sol
IPozBenefit.sol
MainCoinManager.sol
Manageable.sol
Pools.sol
PoolsData.sol
PozBenefit.sol
ThePoolz.sol
TokenList.sol

There were 11 issues found, 0 of which were deemed to be 'critical', and 3 of which were rated as 'high'.

The audit is then followed by a second review of the code at commit **#e895a7376ce31458b5df4c1584dc9eca8b464ccd**. All the issues found by the audit were reviewed and discussed between the development team and the auditing team.

Arcadia also reviewed the updated code at commit #e895a7376ce31458b5df4c1584dc9eca8b464ccd.

- **Issues PLZ 1-6 are fixed by the team in the updated code.**
- **Issue PLZ-7 is still present in the code. However, as discussed with the team for Issue PLZ-7, TokenFilterOn will be used with cautions from the team so that it does not affect the safety and security of the contracts.**
- **Recommendations PLZ-8 and PLZ-9 are fixed by the team.**
- **Recommendations PLZ-10 and PLZ-11 are still present in the code but it does not affect the safety and security of the contracts.**

Severity Rating	Number Of Original Occurrences	Number Of Remaining Occurrences
Critical	0	0
High	3	0
Medium	4	1
Low	0	0
Notice	4	2
Informational	0	0

Arcadia completed the reviews 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.

Findings

1. Verify Transferred Amount of Token

- PLZ-1
- Severity: Medium
- Likelihood: Medium
- Impact: Medium
- Target: ERC20Helper.sol
- Category: Transfer Token
- Finding Type: Dynamic
- Lines: 19-26

Function `TransferToken` should verify that the amount of transferred token should be _amount. Some malicious tokens make malicious token `transfer` out more than what the function expects to send.

```
function TransferToken(
      address _Token,
      address _Reciver,
      uint256 _ammount
  ) internal {
      emit TransferOut(_ammount, _Reciver, _Token);
      ERC20(_Token).transfer(_Reciver, _ammount);
  }
```

Action Recommended: Verify that the max transferred amount in the function should be _amount. Here's one suggestion:

```
function TransferToken(
      address _Token,
      address _Reciver,
      uint256 _ammount
  ) internal {
      emit TransferOut(_ammount, _Reciver, _Token);
      uint256 balBefore =  ERC20(_Token).balanceOf(address(this));
      ERC20(_Token).transfer(_Reciver, _ammount);
      uint256 balAfter =  ERC20(_Token).balanceOf(address(this));
      require(balAfter.sub(balBefore) <= _amount);
  }
```

2. Math is incorrect

- PLZ-2
- Severity: High
- Impact: High

- Target: Invest.sol
- Category: Computation
- Finding Type: Dynamic
- Lines: 49-52, 82-85

`SafeMath` solidity library works for integer number types, e.g. uint256. Due to bias in the `divide` operator for integers, an expression involving `multiply` and `divide` operators should execute the `multiply` operator first.

```
    uint256 EthMinusFee = SafeMath.mul(
            SafeMath.div(msg.value, 10000),
            SafeMath.sub(10000, CalcFee(_PoolId))
        );

    uint256 RegularFeePay = SafeMath.mul(
            SafeMath.div(_Amount, 10000),
            CalcFee(_PoolId)
        );
```

Action Recommended: Multiplication should be done before division. Following is a suggestion.

3. Check for deflationary tokens or tokens with burns on transfers

- PLZ-3
- Severity: High
- Likelihood: High
- Impact: High

- Target: Invest.sol
- Category: Deflationary token sale
- Finding Type: Dynamic
- Lines 58-97

In a deflationary token or a token with burns on transfers, the actual received token amount by calling `transferFrom` function of the token contract would be less than the expected received token amount. In the function `InvestERC20` of the contract `Invest`, token is transferred from the token sale source to the pool contract. However, there is no check for how much the actual token amount the pool receives due to calling the function `TransferInToken`.

```
function InvestERC20(uint256 _PoolId, uint256 _Amount)
      external
      whenNotPaused
  {
      require(_PoolId < poolsCount, "Wrong pool id, InvestERC20 fail");
      require(
          pools[_PoolId].Maincoin != address(0x0),
          "Pool is for ETH, use InvetETH"
      );
      require(_Amount > 10000, "Need invest more then 10000");
      TransferInToken(pools[_PoolId].Maincoin, msg.sender, _Amount);
      uint256 ThisInvestor = NewInvestor(msg.sender, _Amount, _PoolId);
      uint256 Tokens = CalcTokens(_PoolId, _Amount, msg.sender);

      if (pools[_PoolId].IsLocked) {
          Investors[ThisInvestor].TokensOwn = SafeMath.add(
              Investors[ThisInvestor].TokensOwn,
              Tokens
          );
      } else {
          // not locked, will transfer the tokens
          TransferToken(pools[_PoolId].Token, msg.sender, Tokens);
      }

      uint256 RegularFeePay = SafeMath.mul(
          SafeMath.div(_Amount, 10000),
          CalcFee(_PoolId)
      );
      uint256 RegularPaymentMinusFee = SafeMath.sub(_Amount, RegularFeePay);
      FeeMap[pools[_PoolId].Maincoin] = SafeMath.add(
          FeeMap[pools[_PoolId].Maincoin],
```

```
            RegularFeePay
        );
        TransferToken(
            pools[_PoolId].Maincoin,
            pools[_PoolId].Creator,
            RegularPaymentMinusFee
        ); // send money to project owner - the fee stays on contract
        RegisterInvest(_PoolId, Tokens);
    }
```

Action Recommended: Verify the actual token amount the pool receives after calling `TransferInToken` function. Here's a suggestion.

```
uint256 balBefore = ERC20(pools[_PoolId].Maincoin).balanceOf(address(this));
TransferInToken(pools[_PoolId].Maincoin, msg.sender, _Amount);
uint256 balAfter = ERC20(pools[_PoolId].Maincoin).balanceOf(address(this));
_Amount = balAfter.sub(balBefore);
```

4. No timing check for token sale

- PLZ-4
- Severity: High
- Likelihood: High
- Impact: High

- Target: Invest.sol
- Category: Token Sale Open Time
- Finding Type: Dynamic
- Lines: 29-34, 50-62

In the current contract `Invest`, any user can buy tokens before the token sale opening time `OpenForAll`. Functions InvestETH and InvestERC20 should check if the timestamp is greater than OpenForAll so that users cannot buy before the sale open time.

```
function InvestETH(uint256 _PoolId)
        external
        payable
        ReceivETH(msg.value, msg.sender)
        whenNotPaused
    {

...

function InvestERC20(uint256 _PoolId, uint256 _Amount)
        external
        whenNotPaused
    {
```

Action Recommended: Add `require` statements to the beginning of the functions to not allow investing if `block.timestamp < OpenForAll` timestamp.

5. Add reentrancy safeguard to function WithdrawInvestment

- PLZ-5
- Severity: Medium
- Likelihood: Medium
- Impact: Medium

- Target: InvestorData.sol
- Category: Reentrancy safeguard
- Finding Type: Dynamic
- Lines: 15-30

```
function WithdrawInvestment(uint256 _id) public returns (bool) {
        if (IsReadyWithdrawInvestment(_id)) {
            TransferToken(
                pools[Investors[_id].Poolid].Token,
                Investors[_id].InvestorAddress,
                Investors[_id].TokensOwn
            );
            pools[Investors[_id].Poolid].UnlockedTokens = SafeMath.add(
                pools[Investors[_id].Poolid].UnlockedTokens,
                Investors[_id].TokensOwn
            );
            Investors[_id].TokensOwn = 0;
            return true;
        }
        return false;
    }
```

Action Recommended:

- Add reentrancy safegard to the function to make it safe from reentrancy.

6. Check actual token received

- PLZ-6
- Severity: Medium
- Likelihood: Medium
- Impact: Medium
- Target: Pools.sol
- Category: Token Transfer In
- Finding Type: Dynamic
- Lines: 66

In the function `CreatePool`, it should check for the actual received token after the `TransferInToken` function call. This is to avoid receiving less tokens than the expected received token amount due to the token sale itself.

Action Recommended: Refer to Issue PLZ-3 for a recommendation.

7. TokenFilterOn should always be true

- PLZ-7
- Severity: Medium
- Likelihood: Medium
- Impact: Medium

- Target: TokenList.sol
- Category: Filter for special tokens
- Finding Type: Dynamic

`TokenFilterOn` should always be true to avoid attackers adding malicious tokens to the pool system. The system should not allow tokens with constant burn rate for all wallets that are not staked in the token pool (e.g., uniswap pools). Example of this kind of token is `XVIX` where any token that is not staked in a liquidity pool has a constant burn rate of 0.02% per hour.

Having `TokenFilterOn` turned on should also allow the team to filter tokens with rebase features. This is because having rebase, the balance of on-sale token stored in the pool contract can be changed, while the amount of token stored in the pool contract does not change.

Recommendations

8. TestAllowance modifier can be simplified

- PLZ-8
- Severity: Informational
- Impact: Informational

- Target: ERC20Helper.sol
- Category: Informational
- Finding Type: Dynamic

The TestAllowance modifier can be simplified as follows:

```
modifier TestAllownce(
      address _token,
      address _owner,
      uint256 _amount
  ) {
  require(ERC20(_token).allowance(_owner, address(this)) >= _amount, "no allowance");
  _;
  }
```

9. ReceiveETH modifier can be simplified

- PLZ-9
- Severity: Informational
- Impact: Informational
- Target: ETHHelper.sol
- Category: Informational
- Finding Type: Dynamic

The ReceiveETH modifier can be simplified as follows:

```
modifier ReceivETH(uint256 msgValue, address msgSender) {
      require(msgValue >= MinETH, "Send ETH to invest");
      emit TransferInETH(msgValue, msgSender);
      _;
  }
```

10. Make functions GetMyInvestmentIds and GetInvestmentData generic

- PLZ-10
- Severity: Informational
- Impact: Informational
- Target: InvestorData.sol
- Category: Informational
- Finding Type: Dynamic

It is recommended to make functions GetMyInvestmentIds and GetInvestmentData generic so that anyone can check investment of any address, by adding an address-typed parameter to the functions.

11. The current use of SafeMath makes it relatively hard to interpret

- PLZ-11
- Severity: Informational
- Impact: Informational
- Target: All
- Category: Informational
- Finding Type: Dynamic

The current usage of SafeMath makes it relatively hard to interpret and not common practice. For example, the following expression:

```
uint256 EthMinusFee = SafeMath.mul(
          SafeMath.div(msg.value, 10000),
          SafeMath.sub(10000, CalcFee(_PoolId))
      );
```

Can be rewritten as:

```
      uint256 EthMinusFee = msg.value.div(10000).mul(uint256(10000).sub(CalcFee(_PoolId)));
```

The rewritten one is easier to spot the bug in `PLZ-2` where multiplication should be done before division.

Conclusion

Arcadia identified some issues that occurred at git hash #d78a261077b9c5495680b7fd80b73f3cc0f9a1c1 and reviewed the updated code at git hash **#e895a7376ce31458b5df4c1584dc9eca8b464ccd.**

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.