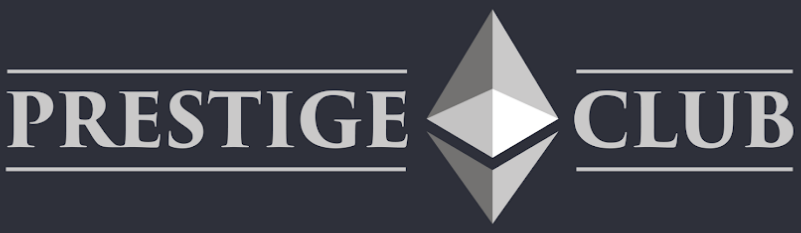
12th JANUARY 2021



SMART CONTRACT AUDIT REPORT

version v2.0

Smart Contract Security Audit and General Analysis

HAECHI AUDIT

Table of Contents

3 Issues (0 Critical, 1 Major, 2 Minor) Found

About HAECHI AUDIT

HAECHI AUDIT is a global leading smart contract security audit and development firm operated by HAECHI LABS. HAECHI AUDIT consists of professionals with years of experience in blockchain R&D and provides the most reliable smart contract security audit and development services.

So far, based on the HAECHI AUDIT's security audit report, our clients have been successfully listed on the global cryptocurrency exchanges such as Huobi, Upbit, OKEX, and others.

Our notable portfolios include SK Telecom, Ground X by Kakao, and Carry Protocol while HAECHI AUDIT has conducted security audits for the world's top projects and enterprises.

Trusted by the industry leaders, we have been incubated by Samsung Electronics and awarded the Ethereum Foundation Grants and Ethereum Community Fund.

Contact : audit@haechi.io

Website : audit.haechi.io

01. Introduction

This report was written to provide a security audit for the Prestige Club smart contract. HAECHI AUDIT conducted the audit focusing on whether Prestige Club smart contract is designed and implemented in accordance with publicly released information and whether it has any security vulnerabilities.

The issues found are classified as CRITICAL, MAJOR, MINOR or TIPS according to their severity.

CRITICAL	Critical issues are security vulnerabilities that MUST be addressed in order to prevent widespread and massive damage.
MAJOR	Major issues contain security vulnerabilities or have faulty implementation issues and need to be fixed.
MINOR	Minor issues are some potential risks that require some degree of modification.
TIPS	Tips could help improve the code's usability and efficiency

HAECHI AUDIT advises addressing all the issues found in this report.

02. Summary

The code used for the audit can be found at Github (https://github.com/HAECHI-LABS/audit-PrestigeClub). The last commit for the code audited is at "ae674814cf3dbd90ab80cbe0a1faebe9be105120".

Issues

HAECHI AUDIT has 0 Critical Issues, 1 Major Issue, and 2 Minor Issues; also, we included 1 Tip category that would improve the usability and/or efficiency of the code.

Severity	Issue	Status
MAJOR	The privilege of the Owner is set relatively high.	(Found - v.1.0) (Resolved - v2.0)
MINOR	A lack of consistency was found between functions that perform similar functions.	(Found - v.1.0) (Acknowledged - v2.0)
MINOR	The value of the token may not be guaranteed during the Token Sell process.	(Found - v.1.0) (Acknowledged - v2.0)
TIPS	When calculating the uint112 value, SafeMath may not be applied.	(Found - v.1.0) (Resolved - v2.0)

03. Overview

Contracts Subject to Audit

- PEth
- PEthDex
- Ownable
- PrestigeClub

Roles

Prestige Club Smart contract has the following authorizations:

- **Owner**

The functions accessible with the authority are as follows.

Role	Functions
Owner	*Ownable#transferOwnership()* *PrestigeClub#setLimits()* *PrestigeClub#setDownlineLimit()* *PrestigeClub#forceSetReferral()* *PrestigeClub#reCalculateImported()* *PrestigeClub#_import()* *PrestigeClub#importUser()*

04. Issues Found

Major : The privilege of the Owner is set relatively high (Found - v.1.0) (Resolved - v2.0)

MAJOR

```
502.  function _import(address[] memory sender, uint112[] memory deposit, address[] memory referer)
      public onlyOwner {
503.      for(uint64 i = 0 ; i < sender.length ; i++){
504.          importUser(sender[i], deposit[i], referer[i]);
505.      }
506.  }
507.
508.  function importUser(address sender, uint112 deposit, address referer) internal onlyOwner {
509.      if(referer != address(0)){
510.          users[referer].referrals.push(sender);
511.          users[sender].referer = referer;
512.      }
513.      uint112 value = deposit;
514.      // Create a position for new accounts
515.      lastPosition++;
516.      users[sender].position = lastPosition;
517.      users[sender].lastPayout = pool_last_draw;
518.      userList.push(sender);
519.
520.      if(referer != address(0)){
521.          updateUpline(sender, referer, value);
522.      }
523.      users[sender].deposit += value;
524.      emit NewDeposit(sender, value);
525.      updateUserPool(sender);
526.      updateDownlineBonusStage(sender);
527.      if(referer != address(0)){
528.          users[referer].directSum += value;
529.          updateUserPool(referer);
530.          updateDownlineBonusStage(referer);
531.      }
532.      depositSum += value;
533.  }
534.
```

Problem Statement

With *PrestigeClub#_import()* function, the owner directly registers a specific address to the club. By using *_import()* function, the owner can register a specific address as *sender* and *referer* simultaneously. When this is repeated, the pertinent address can receive a bonus corresponding to the higher pool only with its own deposit, even without being registered as a *referer* of another person. This is an impossible action in *receive()* function where the user directly registers oneself. Thus, it is determined that the owner's privilege is set relatively high.

Recommendation

Please add a require statement to _import() function that prevents the above situation to reduce the owner's privilege.

Update

[v2.0] - Prestige Club team fixed this issue by adding require statement that prevents the owner to re-import same user.

Minor : A lack of consistency exists between functions that perform similar functions (Found - v.1.0) (Acknowledged - v2.0)

MINOR

```
502.  function _import(address[] memory sender, uint112[] memory deposit, address[] memory referer)
      public onlyOwner {
503.      for(uint64 i = 0 ; i < sender.length ; i++){
504.          importUser(sender[i], deposit[i], referer[i]);
505.      }
506.  }
507.
508.  function importUser(address sender, uint112 deposit, address referer) internal onlyOwner {
509.      if(referer != address(0)){
510.          users[referer].referrals.push(sender);
511.          users[sender].referer = referer;
512.      }
513.      uint112 value = deposit;
514.      // Create a position for new accounts
515.      lastPosition++;
516.      users[sender].position = lastPosition;
517.      users[sender].lastPayout = pool_last_draw;
518.      userList.push(sender);
519.
520.      if(referer != address(0)){
521.          updateUpline(sender, referer, value);
522.      }
523.      users[sender].deposit += value;
524.      emit NewDeposit(sender, value);
525.      updateUserPool(sender);
526.      updateDownlineBonusStage(sender);
527.      if(referer != address(0)){
528.          users[referer].directSum += value;
529.          updateUserPool(referer);
530.          updateDownlineBonusStage(referer);
531.      }
532.      depositSum += value;
533.  }



193.function recieve(uint112 amount) public {
194.      require((users[msg.sender].deposit * 20 / 19) >= minDeposit || amount >= minDeposit, "Mininum
      deposit value not reached");
195.      address sender = msg.sender;
196.      //Transfer peth
197.      peth.transferFrom(sender, address(this), amount);
198.      uint112 value = amount.mul(19).div(20);
199.      bool userExists = users[sender].position != 0;
200.      triggerCalculation();
201.      // Create a position for new accounts
202.      if(!userExists){
203.          lastPosition++;
204.          users[sender].position = lastPosition;
205.          users[sender].lastPayout = (pool_last_draw + 1);
206.          userList.push(sender);
207.      }
208.      address referer = users[sender].referer; //can put outside because referer is always set since
      setReferral() gets called before recieve() in recieve(address)
```

```
209.        if(referer != address(0)){
210.            updateUpline(sender, referer, value);
211.        }
212.        //Update Payouts
213.        if(userExists){
214.            updatePayout(sender);
215.        }
216.        users[sender].deposit = users[sender].deposit.add(value);
217.        //Transfer fee
218.        peth.transfer(owner(), (amount - value));
219.        emit NewDeposit(sender, value);
220.
221.        updateUserPool(sender);
222.        updateDownlineBonusStage(sender);
223.
224.        if(referer != address(0)){
225.            users[referer].directSum = users[referer].directSum.add(value);
226.            updateUserPool(referer);
227.            updateDownlineBonusStage(referer);
228.        }
229.        depositSum = depositSum + value; //Won´t do an overflow since value is uint112 and
    depositSum 128
230.    }
231.
```

Problem Statement

With *PrestigeClub#_import()* function, the owner directly registers a specific address to the club. On the other hand, with *PrestigeClub#recieve()* function, the user registers oneself to the club. These two functions function similarly. However, using *_import()* function can perform operations that are impossible in *receive()* function. For instance, by using *_import()* function, actions can be performed such as setting an address that has not been registered before as *referer* and registering an already-registered address to the club in duplication.

This is considered to be a lack of overall consistency of the code. Thus, it is advised to fix the code if the above situation is unintended.

Recommendation

Add a repuire statement at the same level as *recieve()* function to *_import()* function.

Update

[v2.0] - PrestigeClub team has responded that they are aware of this issue and it is on purpose.

Minor : The value of the token may not be guaranteed during the Token Sell process (Found - v.1.0) (Acknowledged - v2.0)

MINOR

```
47.  function sell(uint256 amount) public {
48.
49.      require(amount > 0, "You need to sell at least some tokens");
50.      require(address(this).balance >= amount, "Not enough Ether to sell amount");
51.
52.      // uint256 allowance = allowance(msg.sender, address(this));
53.      // require(allowance >= amount, "Check the token allowance");
54.
55.      uint256 balance = balanceOf(msg.sender);
56.      require(balance >= amount, "Not enough funds to sell");
57.
58.      // _transfer(msg.sender, address(this), amount);
59.      _burn(msg.sender, amount);
60.
61.      bool success = payable(msg.sender).send(amount);
62.      require(success, "Transfer of ether not successful");
63.      emit Sold(msg.sender, amount);
64.
65.  }
66.
```

Problem Statement

PEthDex#sell() function converts peth token to ether. At this time, the conversion ratio of the two assets is set to 1:1.

At this time, if ether balance is insufficient in the PEthDex contract, the users cannot convert their peth token to ether. Due to this, it is determined that the value of peth token may not be guaranteed.

Update

[v2.0] - PrestigeClub team has responded that they are aware of this issue and it is on purpose.

TIPS : When calculating the uint112 value, OverFlow may occur (Found - v.1.0) (Resolved - v2.0)

TIPS

```
464.  function getDownline() external view returns (uint112, uint){
465.      uint112 sum;
466.
467.      for(uint8 i = 0 ; i < users[msg.sender].downlineVolumes.length ; i++){
468.          sum += users[msg.sender].downlineVolumes[i];
469.      }
470.      uint numUsers = getDownlineUsers(msg.sender);
471.      return (sum, numUsers);
472.  }
473.
```

When calculating the uint112 value in line 468, an integer overflow may occur.

For example, if the sum of downlineVolumes[0] and downlineVolumes[1] of the user exceeds 2^112-1, the maximum value of uint112, a value that differs from the actual calculation result becomes stored in the sum.

Update

[v2.0] - Prestige Club team fixed this issue by adding using safemath when calculating the uint112 value..

05. Disclaimer

This report is not advice on investment, nor does it guarantee the adequacy of a business model and/or a bug-free code. This report should be used only to discuss known technical problems. The code may include problems on Ethereum that are not included in this report. It will be necessary to resolve addressed issues and conduct thorough tests to ensure the safety of the smart contract.

Appendix A. Test Results

The following are the results of a unit test that covers the major logics of the smart contract under audit. The parts in red contain issues and therefore have failed the test.

```
  PEthDex
 #contructor()
  ✓ should mint 1 ether to owner
 #buy()
  ✓ should fail if amountTobuy is 0
  Valid Case
   ✓ owner's ether balance should increase
   ✓ buyer's ether balance should decrease
   ✓ buyer's token balance should increase
   ✓ totalSupply should inncrease
   ✓ should emit Bought event
 #sell()
  ✓ should fail if amountToSell is 0
  ✓ should fail if contract does not have enough ether
  ✓ should fail if sellAmount is larger than balance (40ms)
  ✓ should fail if seller is AddressZero
  Valid Case
   ✓ seller's ether balance should increase
   ✓ contract's ether balance should decrease
   ✓ seller's token balance should decrease
   ✓ totalSupply should decrease
   ✓ should emit Sold event

PEth
 #constructor
  ✓ should store _name = name_
  ✓ should store _symbol = symbol_
  ✓ should store _decimals = 18
 #transfer
  ✓ should fail if recipient is AddressZero
  ✓ should fail if sender's amount is lower than balance
  Valid Case
   ✓ sender's balance should decrease
   ✓ recipient's balance should increase
   ✓ should emit Transfer event
 #approve()
```

```
    ✓ should fail if spender is AddressZero
    valid case
      ✓ allowance should set appropriately
      ✓ should emit Approval event
  #transferFrom()
    ✓ should fail if sender is AddressZero
    ✓ should fail if recipient is AddressZero
    ✓ should fail if sender's amount is lower than transfer amount
    ✓ should fail if allowance is lower than transfer amount
    ✓ should fail even if try to transfer sender's token without approve process
    Valid Case
      ✓ sender's balance should decrease
      ✓ recipient's balance should increase
      ✓ should emit Transfer event
      ✓ allowance should decrease
      ✓ should emit Approval event
  #totalSupply
    ✓ should return totalSupply

PrestigeClub
  #contructor()
    ✓ should store pool_last_draw
  #_import()
    ✓ should fail if msg.sender is not owner
    1) should fail if referer is not imported
    2) should fail if sender = referer
    3) should fail when import same sender again
    valid case
      Case : the first sender does not have referer
        ✓ should write sender's position info
        ✓ should write sender's lastPayout info (39ms)
        ✓ should write userList
        ✓ should increase sender's deposit
        ✓ should emit NewDeposit event
        ✓ should increase deposit sum
      Case : sender does have basic referer
        ✓ should write referer's referrals info (89ms)
        ✓ should write sender's referer info (110ms)
        ✓ should write sender's position info (92ms)
        ✓ should write sender's lastPayout info (100ms)
        ✓ should write userList (106ms)
        ✓ should increase sender's deposit (89ms)
```

```
    ✓ should emit NewDeposit event (90ms)
    ✓ should increase referer's directSum (97ms)
    ✓ should increase referer's pool level (93ms)
    ✓ should emit NewDeposit event (92ms)
    ✓ should increase deposit sum (92ms)
  Case : sender does have complicate referer
    ✓ should write referer's referrals info (131ms)
    ✓ should write sender's referer info (130ms)
    ✓ should write sender's position info (131ms)
    ✓ should write sender's lastPayout info (137ms)
    ✓ should write userList (129ms)
    ✓ should increase sender's deposit (130ms)
    ✓ should emit NewDeposit event (106ms)
    ✓ should increase referer's directSum (125ms)
    ✓ should increase referer's pool level (125ms)
    ✓ should emit PoolReached event (123ms)
    ✓ should emit DownlineBonusStageReached event (109ms)
    ✓ should increase deposit sum (124ms)
    ✓ should emit Payout event (118ms)
#recievex()
  ✓ should fail if msg.sender is not owner
 valid case
  Case : msg.sender is not imported yet
    ✓ contract's balance should increase (63ms)
    ✓ owner's balance should increase (67ms)
    ✓ msg.sender's balance should decrease (66ms)
    ✓ should create position (74ms)
    ✓ msg.sender's deposit should increase (63ms)
    ✓ should increase deposit sum (60ms)
    ✓ should emit NewDeposit event (63ms)
  Case : msg.sender is already imported
    ✓ contract's balance should increase (99ms)
    ✓ owner's balance should increase (98ms)
    ✓ msg.sender's balance should decrease (115ms)
    ✓ should create position (114ms)
    ✓ msg.sender's deposit should increase (92ms)
    ✓ should increase deposit sum (92ms)
    ✓ should emit NewDeposit event (101ms)
    ✓ should emit Payout event
    ✓ should emit DownlineBonusStageReached event (83ms)
    ✓ should emit PoolReached event (85ms)
#recieve()
```

```
  ✓ should fail if msg.sender already have referer
  ✓ should fail if referer is not imported
  ✓ should fail if sender = referer
  ✓ should not set referer when old referer and new referer are same (144ms)
  ✓ should not set referer when msg.sender is older than referer (118ms)
  valid case
    ✓ should add referer's directSum when user has deposit (169ms)
#withdraw()
  ✓ should fail if withdrawal too large amount
  ✓ should fail if withdrawal too small (51ms)
  ✓ should fail if withdrawal amount is smaller than payout
  valid case
    ✓ should decrease msg.sender's payout value
    ✓ should set msg.sender's lastPayedOut (61ms)
    ✓ should decrease contract's ether balance (58ms)
    ✓ should increase msg.sender's ether balance (52ms)
    ✓ should increase owner's ether balance (102ms)
    ✓ should emit Withdraw event (53ms)
#getInterestPayout()
  ✓ should return appropriate payout value
#getDownlinePayout()
  ✓ should return appropriate payout value
#getDirectsPayout()
  ✓ should return appropriate payout value
#reCalculateImported()
  ✓ should fail when msg.sender is not owner
  ✓ should reset user's payout (73ms)
#getDownline()
  4) should return downline info
#getDownlineUsers()
  ✓ should return downline info
#setLimits()
  ✓ should fail when msg.sender is not owner
  ✓ should set appropriate minDeposit
  ✓ should set appropriate minWithdraw
#setDownlineLimit()
  ✓ should fail when msg.sender is not owner
  ✓ should set appropriate minWithdraw
#calculateDirects()
  ✓ should return direct info
```

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/					
Context.sol	**100**	**100**	**100**	**100**	
Dex.sol	**100**	**100**	**100**	**100**	
IERC20.sol	**100**	**100**	**100**	**100**	
Peth.sol	**100**	**100**	**100**	**100**	
PrestigeClubv2.sol	**100**	**100**	**100**	**100**	
contracts/librarise/					
SafeMath.sol	**100**	**100**	**100**	**100**	
SafeMath112.sol	**100**	**100**	**100**	**100**	

[Table 1] Test Case Coverage