// HALBORN

PERA.FINANCE

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **March 8, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	03/08/2021	Gabi Urrutia
0.2	Document Edits	03/12/2021	Gokberk Gulgun
1.0	Final Version	03/15/2021	Gabi Urrutia
1.1	Remediation Plan	03/21/2021	Gokberk Gulgun

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Gokberk Gulgun	Halborn	Gokberk.Gulgun@halborn.com

EXECUTIVE SUMMARY

1.1 INTRODUCTION

Pera.Finance engaged Halborn to conduct a security assessment on their Smart contracts beginning on March 8th, 2021 and ending March 12th, 2021. The security assessment was scoped to the contract PERA.sol and an audit of the security risk and implications regarding the changes introduced by the development team at Pera.Finance prior to its production release shortly following the assessments deadline.

The most important security findings were the use of a low key management policy and the need to correctly implement a Role-Based Access Control Policy. The past attack on PAID network has shown once again that is essential to implement a key management policy based on multi signature to perform critical actions such as deploying or upgrading. In addition, applying the principle of least privilege in privileged accounts allows that if a private key is compromised, the rest of the critical actions can be performed by other privileged users and quickly recovering the control of the contract. Moreover, the contract does NOT contain any obvious exploitation vectors that Halborn was able to leverage within the time frame of testing allotted.

Though the outcome of this security audit is that the PERA team needs to solve some critical issues. Due to scope constraints, only testing and verification of essential properties related to the Liquidity Contract was performed to achieve objectives and deliverables set in the scope.

Halborn recommends performing further testing to validate extended safety and correctness in context to the whole set of contracts. External threats, such as economic attacks, oracle attacks, and inter-contract functions and calls should be validated for expected logic and state.

1.2 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the smart contract audit.While manual testing is recommended to uncover flaws in logic, process,and implementation; automated testing techniques help enhance coverage of smart contracts and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose.
- Smart Contract manual code read and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions(solgraph)
- Manual Assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Scanning of solidity files for vulnerabilities, security hotspots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions.(Slither)
- Testnet deployment (Truffle,Ganache)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident, and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. It's quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that was used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.3 SCOPE

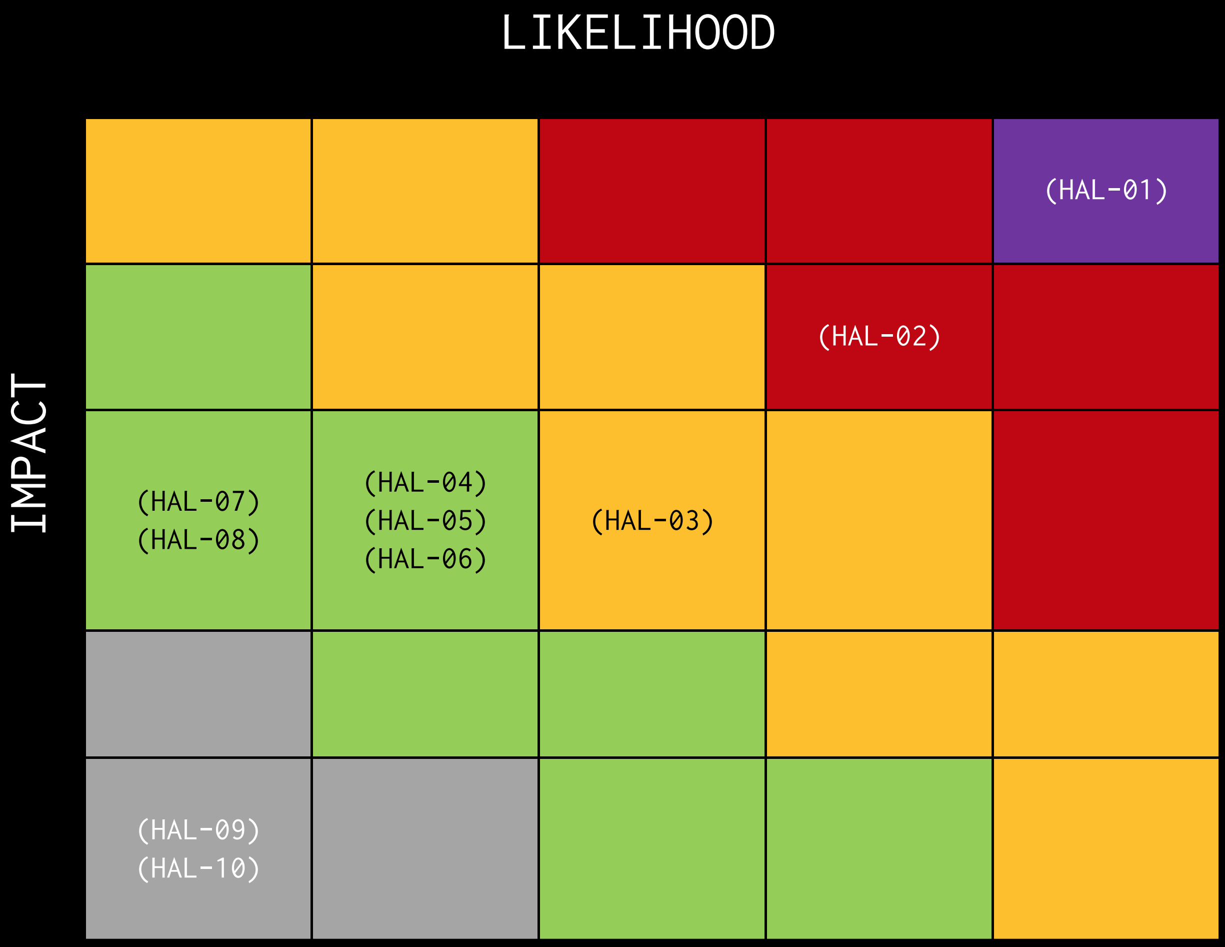
IN-SCOPE:
Code related to pera.sol smart contract.

Specific commit of contract:

e025ab16a39fdaa2e420c9ea95fb496d3c8c92d3

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
1	1	1	5	2


LIKELIHOOD
IMPACT
(HAL-01)
(HAL-02)
(HAL-07)
(HAL-08)
(HAL-04)
(HAL-05)
(HAL-06)
(HAL-03)
(HAL-09)
(HAL-10)

EXECUTIVE SUMMARY

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
IMPROPER KEY MANAGEMENT POLICY	Critical	SOLVED: 04/14/2021
IMPROPER ROLE-BASED ACCESS CONTROL POLICY	High	SOLVED: 04/14/2021
NO TEST COVERAGE	Medium	UNCOVERED: 05/08/2021
FLOATING PRAGMA	Low	SOLVED: 04/14/2021
POSSIBLE RE-ENTRANCY	Low	SOLVED: 04/14/2021
DIVIDE BEFORE MULTIPLY	Low	SOLVED: 04/14/2021
PRAGMA VERSION	Informational	SOLVED: 04/14/2021
FOR LOOP OVER DYNAMIC ARRAY	Informational	SOLVED: 04/14/2021
POSSIBLE MISUSE OF PUBLIC FUNCTIONS	Informational	SOLVED: 04/14/2021
DOCUMENTATION	Informational	SOLVED: 05/08/2021
STATIC ANALYSIS	-	-
AUTOMATED SECURITY SCAN RESULTS	-	-

FINDINGS & TECH DETAILS

3.1 (HAL-01) IMPROPER KEY MANAGEMENT POLICY - CRITICAL

Description:

A fundamental principle of blockchain is decentralization which should be applied as widely as possible in all areas, including key management. Using a single private key to manage a smart contract and perform privileged actions such as deploying or upgrading the contract is risky. If the private key is compromised, it could have devastating consequences. For example, on March 5, 2021, the PAID Network smart contract was successfully attacked despite the smart-contract being previously audited. Approximately $100 million of PAID tokens were extracted by the attacker. In that case, the private key was compromised and the attacker upgraded and replaced the original smart contract with a malicious version that allowed tokens to be burned and minted. Had best practices been implemented in the key management policy, the attacker could not have upgraded the contract using a single private key. Requiring multiple signatures in the key-management policy prevents a single user from performing any critical actions.

Reference: https://halborn.com/explained-the-paid-network-hack-march-2021/

Risk Level:

Likelihood - 5
Impact - 5

Recommendations:

Require multiple signatures in the key-management policy to avoid a private-key compromise resulting in loss of control over the smart contract.

Remediation Plan:

SOLVED: Pera.Finance Team will use a multi-signature wallet for the deployment to the mainnet.

3.2 (HAL-02) IMPROPER ROLE-BASED ACCESS CONTROL POLICY - HIGH

Description:

Implementing a valid access control policy is an essential step in maintaining the security of a smart contract. Access to smart-contract features, such as minting or burning tokens, and pausing contracts, are protected by access control. For instance, ownership is the most common form of access control. By default, the owner of a contract (the account that deployed it) can perform administrative tasks on the contract. Additional authorization levels are needed to implement the least privilege principle, also known as least-authority, which ensures only authorized processes, users, or programs can access the necessary resources or information. The ownership role is helpful in a simple system, but more complex projects require more roles by using role-based access control.

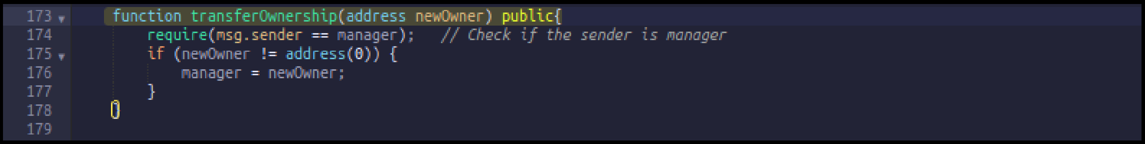
There should be multiple roles such as manager, minter, admin, or pauser in contracts that use a proxy contract. In the pera.sol contract, manager is the only privileged role. Manager can transfer the contract ownership, include/exclude accounts and call addLPToken() function.

In conclusion, the manager role can perform too many privileged actions in the PERA smart contract. If the private key of the manager account is compromised and multi-signature was not implemented, the attacker can perform many actions such as transferring ownership or whitelisting/blacklisting the contract without following the principle of least privilege.

Code Location:

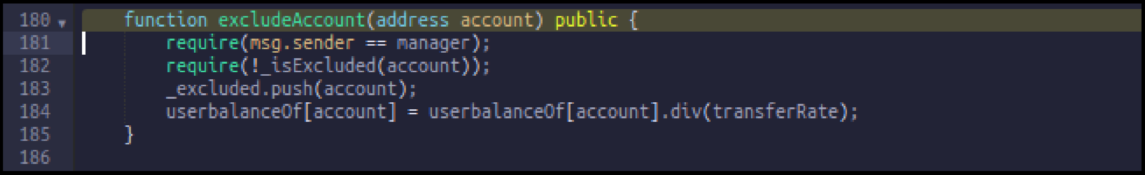
Manager role can access below functions:

pera.sol Lines #173-179



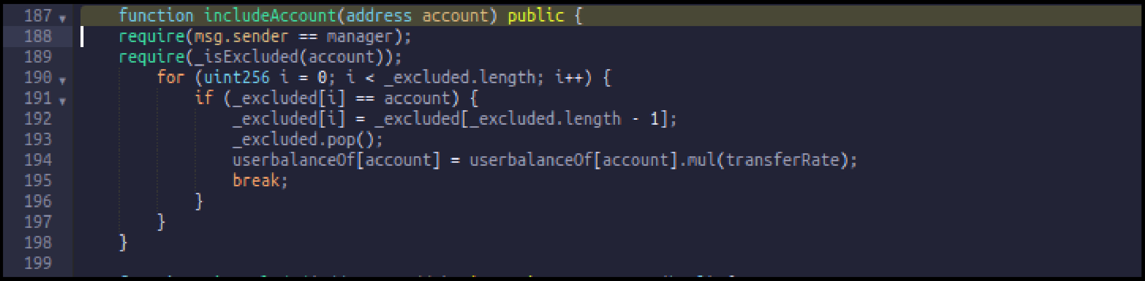
```
173   function transferOwnership(address newOwner) public{
174       require(msg.sender == manager);   // Check if the sender is manager
175       if (newOwner != address(0)) {
176           manager = newOwner;
177       }
178   }
179
```

pera.sol Lines #180-186



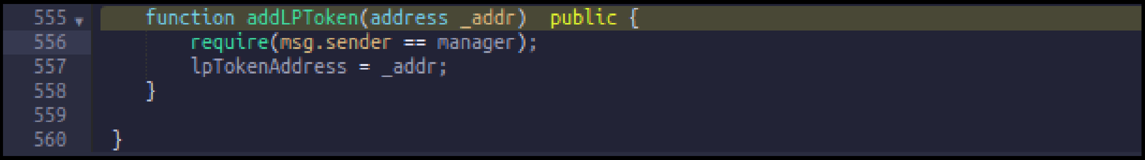
```
180   function excludeAccount(address account) public {
181       require(msg.sender == manager);
182       require(!_isExcluded(account));
183       _excluded.push(account);
184       userbalanceOf[account] = userbalanceOf[account].div(transferRate);
185   }
186
```

pera.sol Lines #187-199



```
187   function includeAccount(address account) public {
188   require(msg.sender == manager);
189   require(_isExcluded(account));
190       for (uint256 i = 0; i < _excluded.length; i++) {
191           if (_excluded[i] == account) {
192               _excluded[i] = _excluded[_excluded.length - 1];
193               _excluded.pop();
194               userbalanceOf[account] = userbalanceOf[account].mul(transferRate);
195               break;
196           }
197       }
198   }
199
```

pera.sol Lines #555-560



```
555       function addLPToken(address _addr)  public {
556           require(msg.sender == manager);
557           lpTokenAddress = _addr;
558       }
559
560   }
```

Listing 1

```
function transferOwnership(address newOwner) public
function excludeAccount(address account) public
function includeAccount(address account) public
function addLPToken(address _addr)  public
```

Risk Level:

Likelihood - 4
Impact - 4

Recommendation:

We recommend using role-based access control, based upon the principle of least privilege, to lock permissioned functions using different roles.

Reference:https://www.cyberark.com/what-is/least-privilege/

Remediation Plan:

SOLVED: Pera.Finance Team will use a multi-signature wallet for the deployment to the mainnet.

3.3 (HAL-03) NO TEST COVERAGE - MEDIUM

Risk Level:

Likelihood - 3
Impact - 3

Description:

Unlike traditional software, smart contracts can not be modified unless deployed using a proxy contract. Because of the permanence, unit tests and functional testing are recommended to ensure the code works correctly before deployment. Mocha and Chai are valuable tools to perform unit tests in smart contracts. Mocha is a Javascript testing framework for creating synchronous and asynchronous unit tests, and Chai is a library with assertion functionality such as assert or expect and should be used to develop custom unit tests.

References:
https://github.com/mochajs/mocha
https://github.com/chaijs/chai
https://docs.openzeppelin.com/learn/writing-automated-tests

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

We recommend performing as many test cases as possible to cover all conceivable scenarios in the smart contract.

Remediation Plan:

UNCOVERED: Pera.Finance Team is prepared manual test coverage documentation.

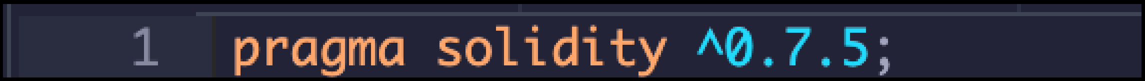
3.4 (HAL-04) FLOATING PRAGMA - LOW

Description:

The Pera.Finance contract uses a floating pragma ˆ0.7.5. Contracts should be deployed with the same compiler version and flags with which they have been tested. Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, either an outdated compiler version that might introduce bugs that negatively affect the contract system or a pragma version too new that has not been extensively tested.

Code Location:

pera.sol Line #~1



```
pragma solidity ^0.7.5;
```

Reference: https://consensys.github.io/smart-contract-best-practices/recommendations/#lock-pragmas-to-specific-compiler-version

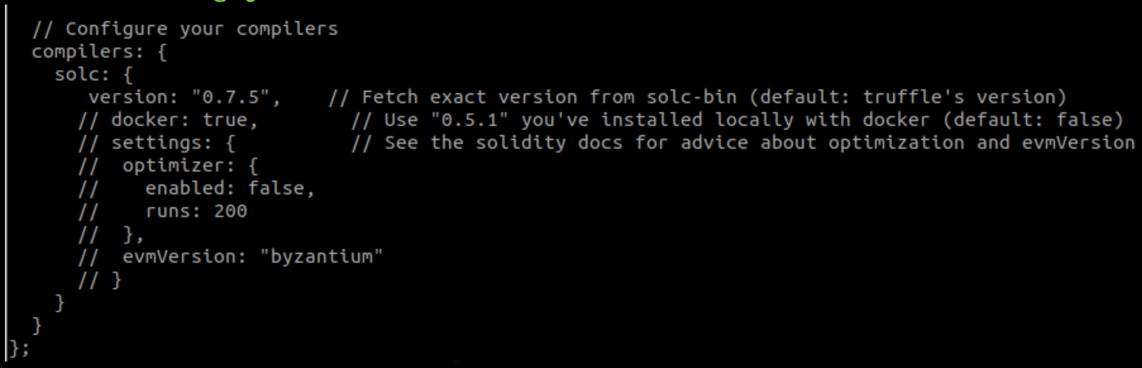
Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

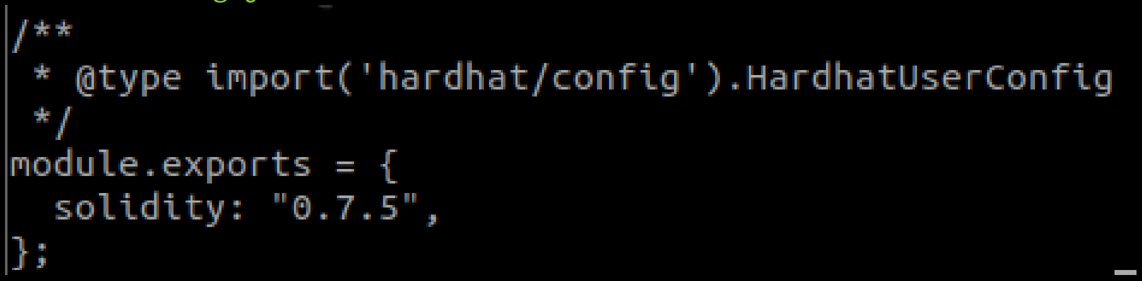
Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment. The pragma can be locked in the code by removing the caret (ˆ) and by specifying the version in the Truffle configuration file truffle-config.js or hardhat.config.js if using the HardHat framework.

truffle-config.js



```
  // Configure your compilers
  compilers: {
    solc: {
       version: "0.7.5",    // Fetch exact version from solc-bin (default: truffle's version)
      // docker: true,        // Use "0.5.1" you've installed locally with docker (default: false)
      // settings: {          // See the solidity docs for advice about optimization and evmVersion
      //  optimizer: {
      //    enabled: false,
      //    runs: 200
      //  },
      //  evmVersion: "byzantium"
      // }
    }
  }
};
```

hardhat.config.js



```
/**
 * @type import('hardhat/config').HardhatUserConfig
 */
module.exports = {
  solidity: "0.7.5",
};
```

Remediation Plan:

SOLVED: Pera.Finance Team locked Pragma Version to 0.6.12

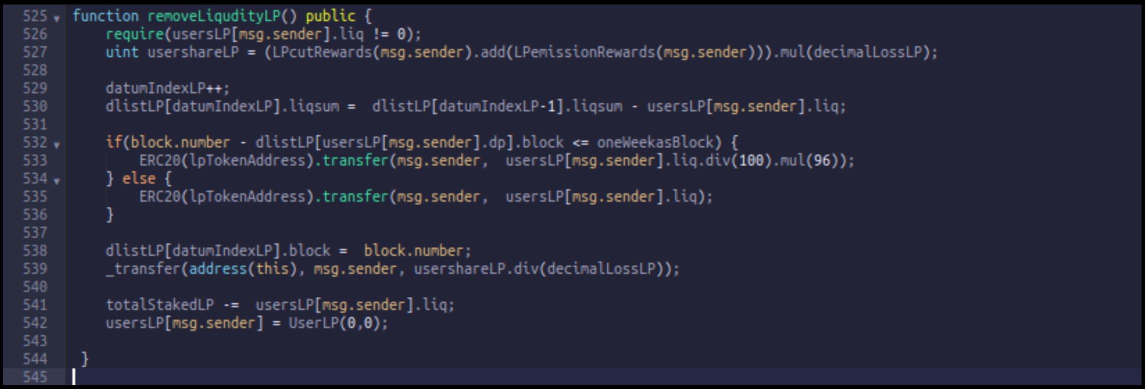
3.5 (HAL-05) POSSIBLE RE-ENTRANCY - LOW

Description:

Calling external contracts is dangerous if functions or variables are used after the external call. The transfer function is executed before checking the totalRewards value, potentially allowing an attacker to hijack the control flow using an external contract implementing a recursive call.

Code Location:

contract.sol Line #526-545



```
525 function removeLiqudityLP() public {
526     require(usersLP[msg.sender].liq != 0);
527     uint usershareLP = (LPcutRewards(msg.sender).add(LPemissionRewards(msg.sender))).mul(decimalLossLP);
528 
529     datumIndexLP++;
530     dlistLP[datumIndexLP].liqsum =  dlistLP[datumIndexLP-1].liqsum - usersLP[msg.sender].liq;
531 
532     if(block.number - dlistLP[usersLP[msg.sender].dp].block <= oneWeekasBlock) {
533         ERC20(lpTokenAddress).transfer(msg.sender,  usersLP[msg.sender].liq.div(100).mul(96));
534     } else {
535         ERC20(lpTokenAddress).transfer(msg.sender,  usersLP[msg.sender].liq);
536     }
537 
538     dlistLP[datumIndexLP].block =  block.number;
539     _transfer(address(this), msg.sender, usershareLP.div(decimalLossLP));
540 
541     totalStakedLP -=  usersLP[msg.sender].liq;
542     usersLP[msg.sender] = UserLP(0,0);
543 
544  }
545 
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

External calls should be at the end of the function to prevent an attacker from taking over the control flow. Check the totalRewards before calling the transfer function. The totalStakedLP and usersLP variables should be called before invoking _transfer(address(this), msg.sender, usershareLP

`.div(decimalLossLP));`.

Remediation Plan:

SOLVED: External calls have been moved to the end by the Pera.Finance Team.

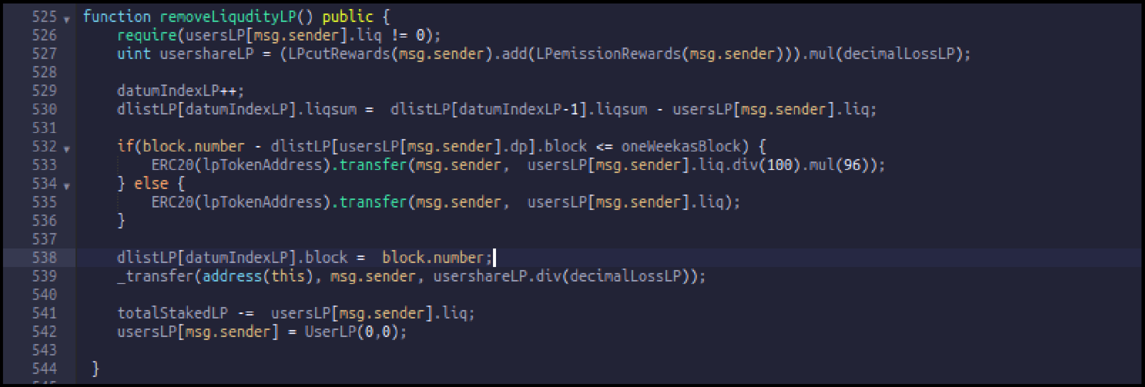
3.6 (HAL-06) DIVIDE BEFORE MULTIPLY - LOW

Description:

In Solidity, integer division might truncate. Performing multiplication before division can sometimes avoid a loss of precision. In this audit, multiple instances were found where division is performed before multiplication in the contract.

Code Location:

contract.sol Line #533



```
525 function removeLiqudityLP() public {
526     require(usersLP[msg.sender].liq != 0);
527     uint usershareLP = (LPcutRewards(msg.sender).add(LPemissionRewards(msg.sender))).mul(decimalLossLP);
528 
529     datumIndexLP++;
530     dlistLP[datumIndexLP].liqsum =  dlistLP[datumIndexLP-1].liqsum - usersLP[msg.sender].liq;
531 
532     if(block.number - dlistLP[usersLP[msg.sender].dp].block <= oneWeekasBlock) {
533         ERC20(lpTokenAddress).transfer(msg.sender,  usersLP[msg.sender].liq.div(100).mul(96));
534     } else {
535         ERC20(lpTokenAddress).transfer(msg.sender,  usersLP[msg.sender].liq);
536     }
537 
538     dlistLP[datumIndexLP].block =  block.number;
539     _transfer(address(this), msg.sender, usershareLP.div(decimalLossLP));
540 
541     totalStakedLP -=  usersLP[msg.sender].liq;
542     usersLP[msg.sender] = UserLP(0,0);
543 
544  }
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

Perform multiplication operations before division to preserve the precision of values for non-floating data types. A sample solution is provided below.

Listing 2

```
ERC20(lpTokenAddress).transfer(msg.sender,  (usersLP[msg.sender].
    liq.mul(96)).div(100));
```

Remediation Plan:

SOLVED: In the updated contract code, the Multiplication operation is performed before division.

3.7 (HAL-07) PRAGMA VERSION - LOW

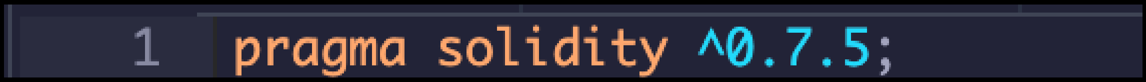
Description:

Pera.Finance contract uses one of the latest pragma version (0.7.5) which was released on November 18, 2020. The latest pragma version (0.8.2) was released in March 2021. Many pragma versions have been lately released, going from version 0.6.x to the recently released version 0.8.x. in just 6 months.

Reference: https://github.com/ethereum/solidity/releases

Code Location:

pera.sol Line #~1



```
pragma solidity ^0.7.5;
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

In the Solitidy Github repository, there is a json file where are all bugs finding in the different compiler versions. No bugs have been found in > 0.7.3 versions but very few in 0.7.0 -- 0.7.3. So, the latest tested and stable version is pragma 0.6.12. Furthermore, pragma 0.6.12

is widely used by Solidity developers and has been extensively tested in many security audits.

Reference: https://github.com/ethereum/solidity/blob/develop/docs/bugs_by_version.json

SOLVED: Pera.Finance Team locked Pragma Version to 0.6.12

3.8 (HAL-08) FOR LOOP OVER DYNAMIC ARRAY - LOW

Description:

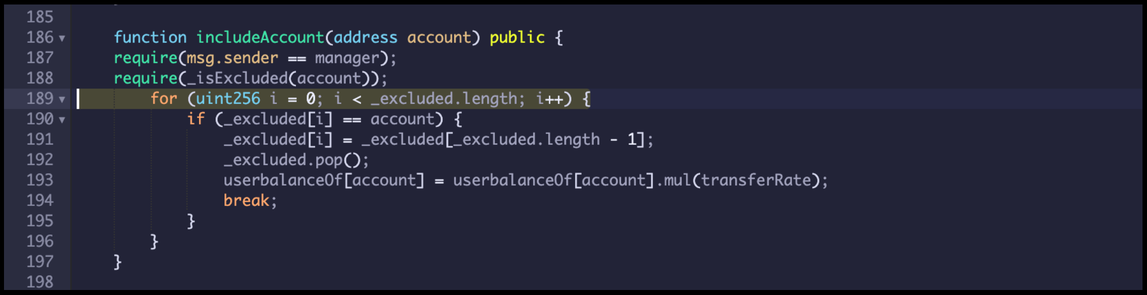
When smart contracts are deployed or functions inside them are called, the execution of these actions always requires a certain amount of gas, based on how much computation is needed to complete them. The Ethereum network specifies a block gas limit and the sum of all transactions included in a block cannot exceed the threshold.

Programming patterns that are harmless in centralized applications can lead to Denial of Service conditions in smart contracts when the cost of executing a function exceeds the block gas limit. Modifying an array of unknown size, that increases in size over time, can lead to such a Denial of Service condition.

A situation in which the block gas limit can be an issue is in sending funds to an array of addresses. Even without any malicious intent, this can easily go wrong. Just by having too large an array of users to pay can max out the gas limit and prevent the transaction from ever succeeding.

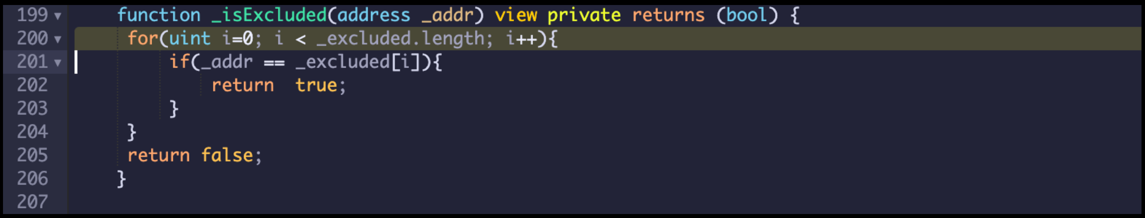
Code Location:

pera.sol Line #~189



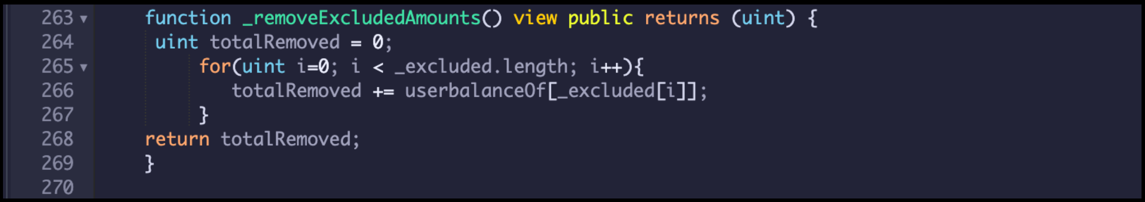
```
185
186     function includeAccount(address account) public {
187     require(msg.sender == manager);
188     require(_isExcluded(account));
189         for (uint256 i = 0; i < _excluded.length; i++) {
190             if (_excluded[i] == account) {
191                 _excluded[i] = _excluded[_excluded.length - 1];
192                 _excluded.pop();
193                 userbalanceOf[account] = userbalanceOf[account].mul(transferRate);
194                 break;
195             }
196         }
197     }
198
```

pera.sol Line #~200



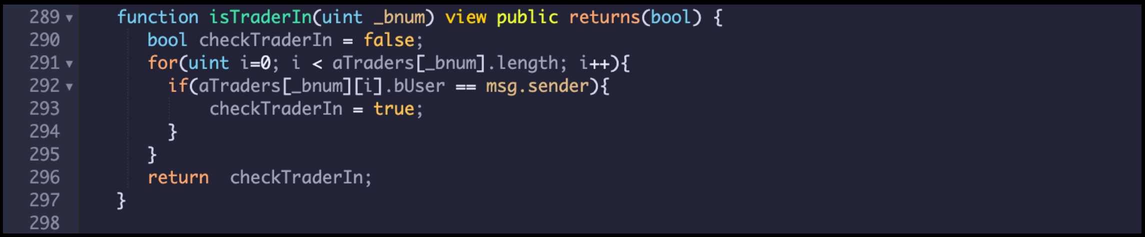
```
199    function _isExcluded(address _addr) view private returns (bool) {
200     for(uint i=0; i < _excluded.length; i++){
201         if(_addr == _excluded[i]){
202             return  true;
203         }
204     }
205     return false;
206    }
207
```

pera.sol Line #~265



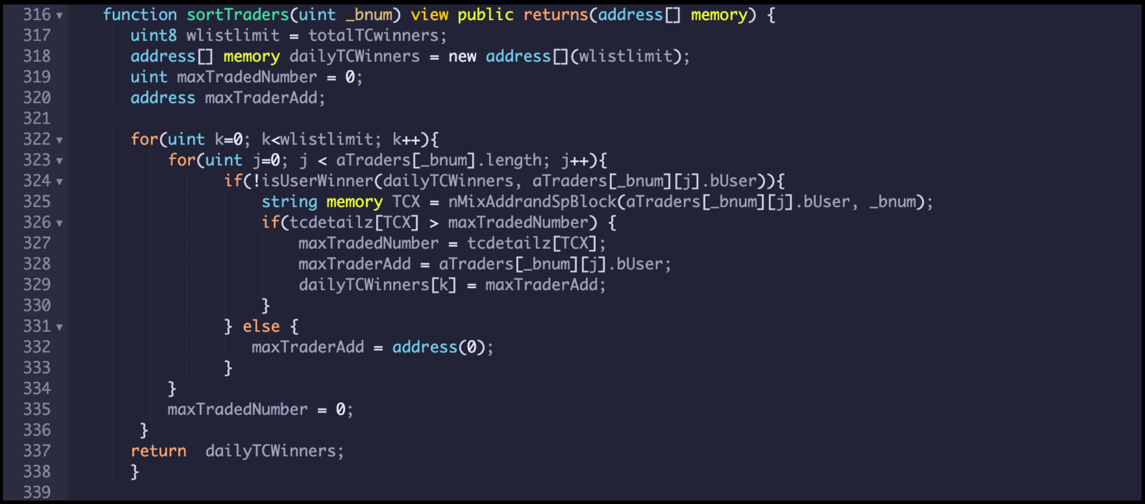
```
263     function _removeExcludedAmounts() view public returns (uint) {
264      uint totalRemoved = 0;
265          for(uint i=0; i < _excluded.length; i++){
266             totalRemoved += userbalanceOf[_excluded[i]];
267          }
268     return totalRemoved;
269     }
270
```

pera.sol Line #~291



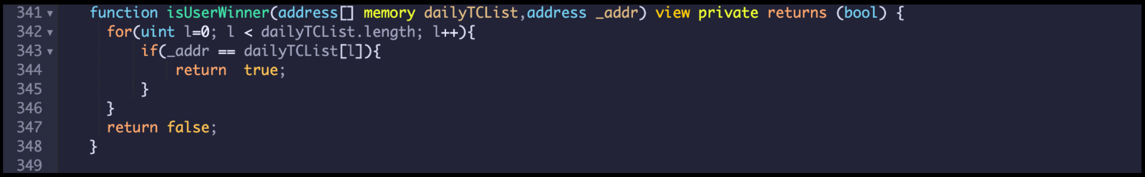
```
289    function isTraderIn(uint _bnum) view public returns(bool) {
290       bool checkTraderIn = false;
291       for(uint i=0; i < aTraders[_bnum].length; i++){
292         if(aTraders[_bnum][i].bUser == msg.sender){
293             checkTraderIn = true;
294         }
295       }
296       return  checkTraderIn;
297    }
298
```

pera.sol Line #~322-323



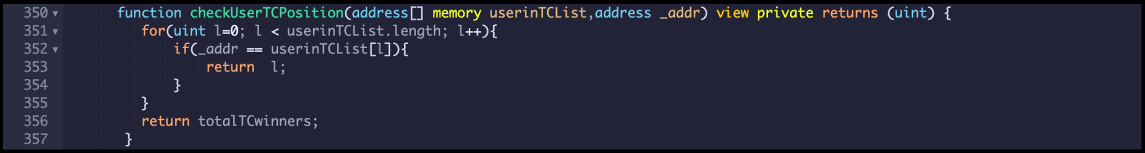
```
316    function sortTraders(uint _bnum) view public returns(address[] memory) {
317       uint8 wlistlimit = totalTCwinners;
318       address[] memory dailyTCWinners = new address[](wlistlimit);
319       uint maxTradedNumber = 0;
320       address maxTraderAdd;
321
322       for(uint k=0; k<wlistlimit; k++){
323           for(uint j=0; j < aTraders[_bnum].length; j++){
324                 if(!isUserWinner(dailyTCWinners, aTraders[_bnum][j].bUser)){
325                     string memory TCX = nMixAddrandSpBlock(aTraders[_bnum][j].bUser, _bnum);
326                     if(tcdetailz[TCX] > maxTradedNumber) {
327                         maxTradedNumber = tcdetailz[TCX];
328                         maxTraderAdd = aTraders[_bnum][j].bUser;
329                         dailyTCWinners[k] = maxTraderAdd;
330                     }
331                 } else {
332                    maxTraderAdd = address(0);
333                 }
334           }
335           maxTradedNumber = 0;
336        }
337       return  dailyTCWinners;
338       }
339
```

pera.sol Line #~342



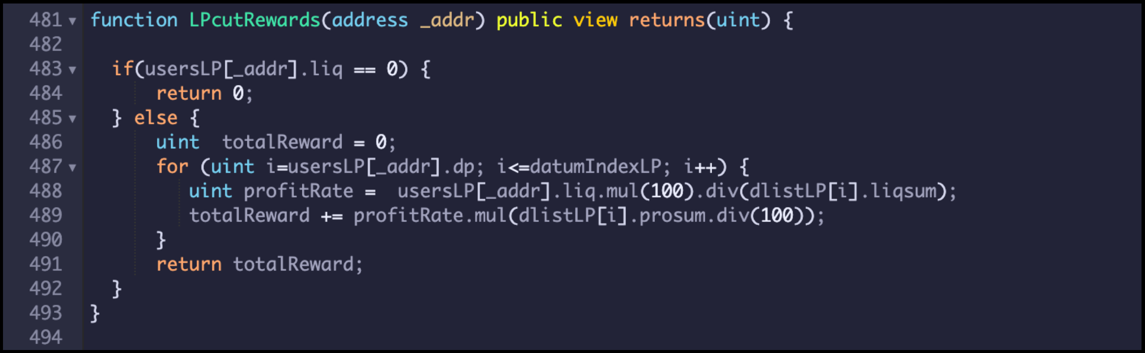
```
341     function isUserWinner(address[] memory dailyTCList,address _addr) view private returns (bool) {
342       for(uint l=0; l < dailyTCList.length; l++){
343           if(_addr == dailyTCList[l]){
344               return  true;
345           }
346       }
347       return false;
348     }
349
```

pera.sol Line #~351



```
350     function checkUserTCPosition(address[] memory userinTCList,address _addr) view private returns (uint) {
351       for(uint l=0; l < userinTCList.length; l++){
352           if(_addr == userinTCList[l]){
353               return  l;
354           }
355       }
356       return totalTCwinners;
357     }
```

pera.sol Line #~487



```
481 function LPcutRewards(address _addr) public view returns(uint) {
482 
483   if(usersLP[_addr].liq == 0) {
484       return 0;
485   } else {
486       uint  totalReward = 0;
487       for (uint i=usersLP[_addr].dp; i<=datumIndexLP; i++) {
488           uint profitRate =  usersLP[_addr].liq.mul(100).div(dlistLP[i].liqsum);
489           totalReward += profitRate.mul(dlistLP[i].prosum.div(100));
490       }
491       return totalReward;
492   }
493 }
494
```

Risk Level:

Likelihood - 1

Impact - 3

Recommendation:

Actions that require looping across the entire data structure should be avoided. If you absolutely must loop over an array of unknown size, then you should plan for it to potentially take multiple blocks, and therefore require multiple transactions.

SOLVED: Pera.Finance Team completed changes to reduce the number of cycles.

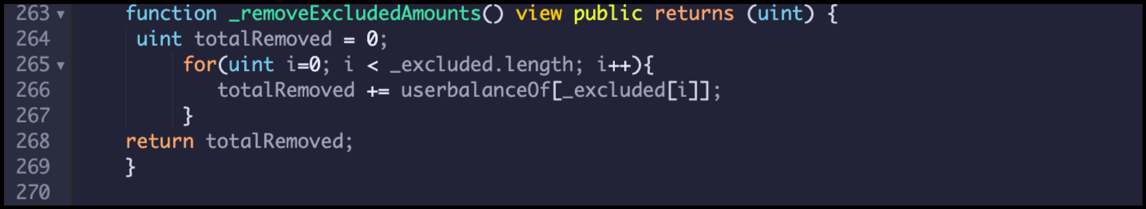
3.9 (HAL-09) POSSIBLE MISUSE OF PUBLIC FUNCTIONS - INFORMATIONAL

Description:

In public functions, array arguments are immediately copied to memory, while external functions can read directly from calldata. Reading call-data is cheaper than memory allocation. Public functions need to write the arguments to memory because public functions may be called internally. Internal calls are passed internally by pointers to memory. Thus, the function expects its arguments being located in memory when the compiler generates the code for an internal function.

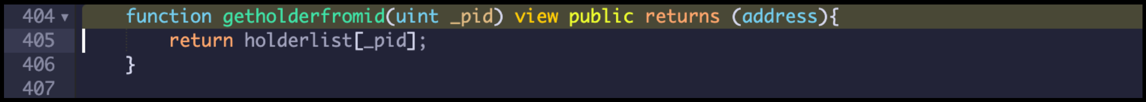
Code Location:

pera.sol Line #~263



```
263     function _removeExcludedAmounts() view public returns (uint) {
264      uint totalRemoved = 0;
265          for(uint i=0; i < _excluded.length; i++){
266              totalRemoved += userbalanceOf[_excluded[i]];
267          }
268     return totalRemoved;
269     }
270
```

pera.sol Line #~404



```
404     function getholderfromid(uint _pid) view public returns (address){
405         return holderlist[_pid];
406     }
407
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider declaring external variables instead of public variables. A best practice is to use external if expecting a function to only be called externally and public if called internally. Public functions are always accessible, but external functions are only available to external callers.

SOLVED: Pera.Finance Team marked functions as an external.

3.10 (HAL-10) DOCUMENTATION - INFORMATIONAL

Description:

The documentation provided by the PERA team is not complete. For instance, the documentation included in the GitHub repository should include a walkthrough to deploy and test the smart contracts.

Recommendation:

Consider updating the documentation in Github for greater ease when contracts are deployed and tested. Have a Non-Developer or QA resource work through the process to make sure it addresses any gaps in the set-up steps due to technical assumptions.

Remediation Plan:

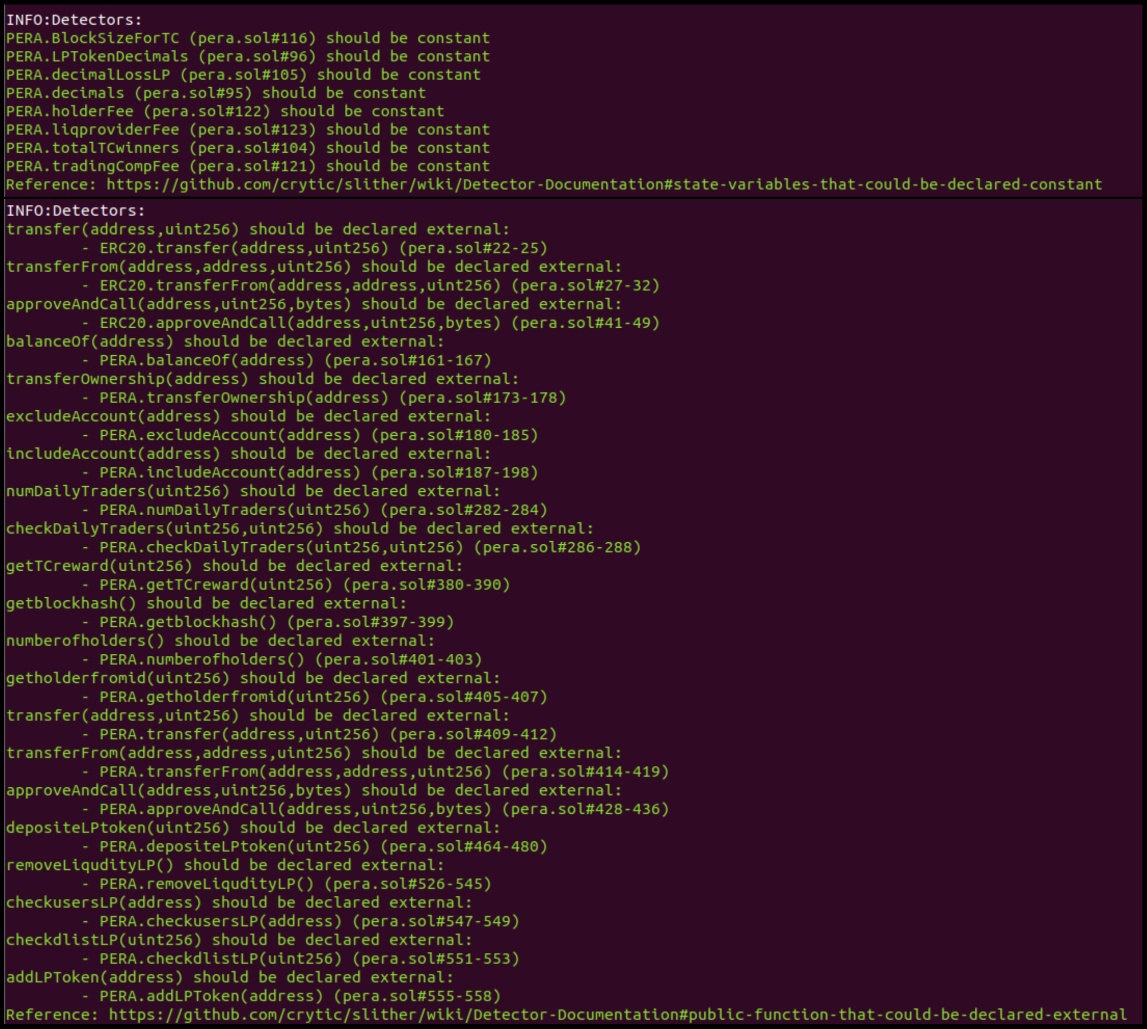
SOLVED: Pera.Finance Team documented all deployment stage.

3.11 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance coverage of certain areas of the scoped contract. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their abi and binary formats, Slither was run on the all-scoped contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire codebase.

Results:



```
INFO:Detectors:
PERA.BlockSizeForTC (pera.sol#116) should be constant
PERA.LPTokenDecimals (pera.sol#96) should be constant
PERA.decimalLossLP (pera.sol#105) should be constant
PERA.decimals (pera.sol#95) should be constant
PERA.holderFee (pera.sol#122) should be constant
PERA.liqproviderFee (pera.sol#123) should be constant
PERA.totalTCwinners (pera.sol#104) should be constant
PERA.tradingCompFee (pera.sol#121) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
```

```
INFO:Detectors:
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (pera.sol#22-25)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (pera.sol#27-32)
approveAndCall(address,uint256,bytes) should be declared external:
        - ERC20.approveAndCall(address,uint256,bytes) (pera.sol#41-49)
balanceOf(address) should be declared external:
        - PERA.balanceOf(address) (pera.sol#161-167)
transferOwnership(address) should be declared external:
        - PERA.transferOwnership(address) (pera.sol#173-178)
excludeAccount(address) should be declared external:
        - PERA.excludeAccount(address) (pera.sol#180-185)
includeAccount(address) should be declared external:
        - PERA.includeAccount(address) (pera.sol#187-198)
numDailyTraders(uint256) should be declared external:
        - PERA.numDailyTraders(uint256) (pera.sol#282-284)
checkDailyTraders(uint256,uint256) should be declared external:
        - PERA.checkDailyTraders(uint256,uint256) (pera.sol#286-288)
getTCreward(uint256) should be declared external:
        - PERA.getTCreward(uint256) (pera.sol#380-390)
getblockhash() should be declared external:
        - PERA.getblockhash() (pera.sol#397-399)
numberofholders() should be declared external:
        - PERA.numberofholders() (pera.sol#401-403)
getholderfromid(uint256) should be declared external:
        - PERA.getholderfromid(uint256) (pera.sol#405-407)
transfer(address,uint256) should be declared external:
        - PERA.transfer(address,uint256) (pera.sol#409-412)
transferFrom(address,address,uint256) should be declared external:
        - PERA.transferFrom(address,address,uint256) (pera.sol#414-419)
approveAndCall(address,uint256,bytes) should be declared external:
        - PERA.approveAndCall(address,uint256,bytes) (pera.sol#428-436)
depositeLPtoken(uint256) should be declared external:
        - PERA.depositeLPtoken(uint256) (pera.sol#464-480)
removeLiqudityLP() should be declared external:
        - PERA.removeLiqudityLP() (pera.sol#526-545)
checkusersLP(address) should be declared external:
        - PERA.checkusersLP(address) (pera.sol#547-549)
checkdlistLP(uint256) should be declared external:
        - PERA.checkdlistLP(uint256) (pera.sol#551-553)
addLPToken(address) should be declared external:
        - PERA.addLPToken(address) (pera.sol#555-558)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

According to the test results, most of the findings found by slither were considered as false positives. Relevant findings were reviewed by the auditors.

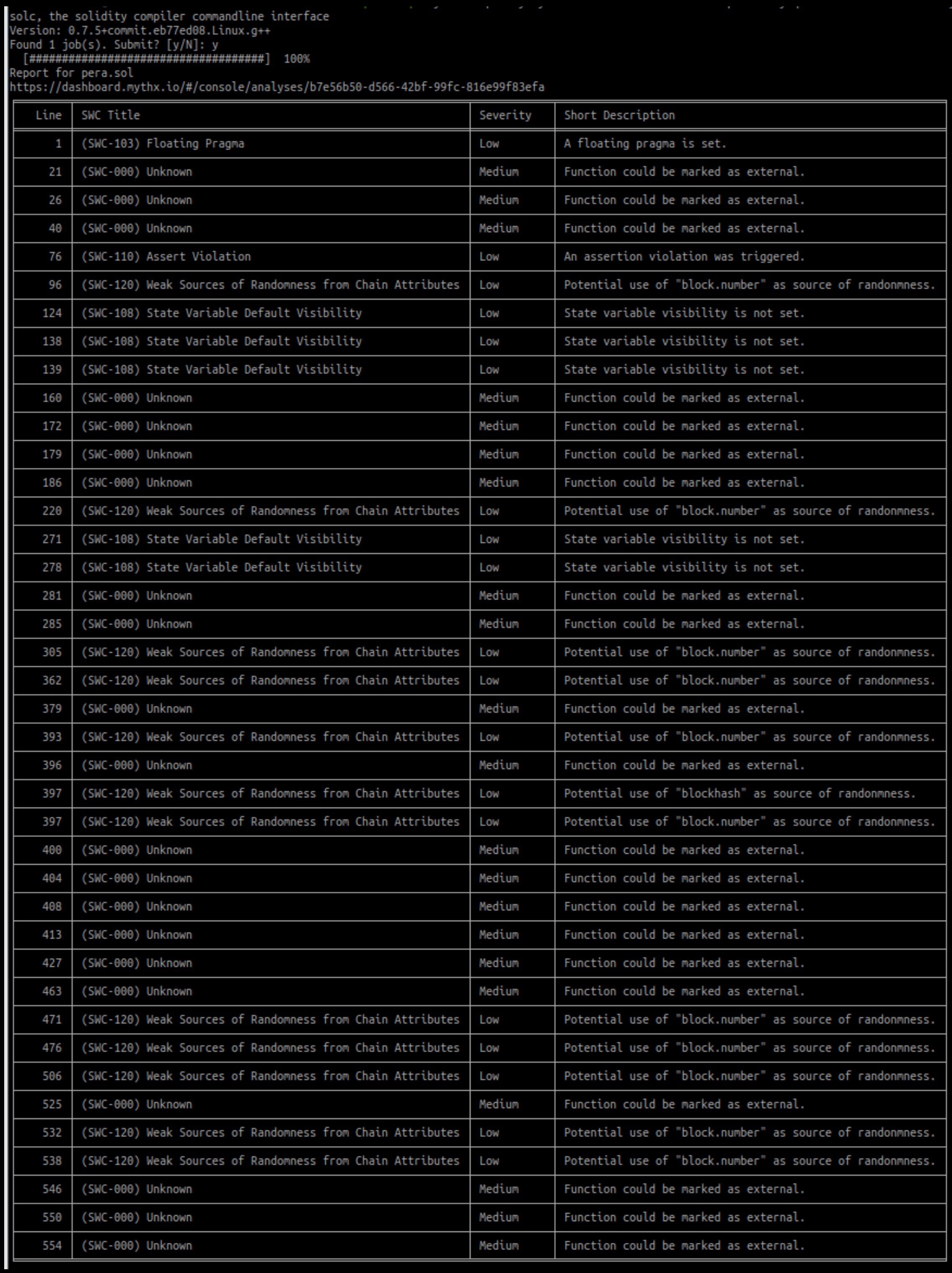
3.12 AUTOMATED SECURITY SCAN RESULTS

Description:

Halborn used automated security scanners to assist with detection of well known security issues, and identify low-hanging fruit on the scoped contract targeted for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on the testers machine, and sent the compiled results to MythX to locate any vulnerabilities. Only security findings were considered in-scope, and the analysis was limited to issues in the PERA.sol contract.

Results:

All relevant findings were founded in the manual code review.



```
solc, the solidity compiler commandline interface
Version: 0.7.5+commit.eb77ed08.Linux.g++
Found 1 job(s). Submit? [y/N]: y
  [####################################]  100%
Report for pera.sol
https://dashboard.mythx.io/#/console/analyses/b7e56b50-d566-42bf-99fc-816e99f83efa
| Line | SWC Title                                            | Severity | Short Description                                     |
|    1 | (SWC-103) Floating Pragma                            | Low      | A floating pragma is set.                             |
|   21 | (SWC-000) Unknown                                    | Medium   | Function could be marked as external.                 |
|   26 | (SWC-000) Unknown                                    | Medium   | Function could be marked as external.                 |
|   40 | (SWC-000) Unknown                                    | Medium   | Function could be marked as external.                 |
|   76 | (SWC-110) Assert Violation                           | Low      | An assertion violation was triggered.                 |
|   96 | (SWC-120) Weak Sources of Randomness from Chain Attributes | Low | Potential use of "block.number" as source of randonmness. |
|  124 | (SWC-108) State Variable Default Visibility          | Low      | State variable visibility is not set.                 |
|  138 | (SWC-108) State Variable Default Visibility          | Low      | State variable visibility is not set.                 |
|  139 | (SWC-108) State Variable Default Visibility          | Low      | State variable visibility is not set.                 |
|  160 | (SWC-000) Unknown                                    | Medium   | Function could be marked as external.                 |
|  172 | (SWC-000) Unknown                                    | Medium   | Function could be marked as external.                 |
|  179 | (SWC-000) Unknown                                    | Medium   | Function could be marked as external.                 |
|  186 | (SWC-000) Unknown                                    | Medium   | Function could be marked as external.                 |
|  220 | (SWC-120) Weak Sources of Randomness from Chain Attributes | Low | Potential use of "block.number" as source of randonmness. |
|  271 | (SWC-108) State Variable Default Visibility          | Low      | State variable visibility is not set.                 |
|  278 | (SWC-108) State Variable Default Visibility          | Low      | State variable visibility is not set.                 |
|  281 | (SWC-000) Unknown                                    | Medium   | Function could be marked as external.                 |
|  285 | (SWC-000) Unknown                                    | Medium   | Function could be marked as external.                 |
|  305 | (SWC-120) Weak Sources of Randomness from Chain Attributes | Low | Potential use of "block.number" as source of randonmness. |
|  362 | (SWC-120) Weak Sources of Randomness from Chain Attributes | Low | Potential use of "block.number" as source of randonmness. |
|  379 | (SWC-000) Unknown                                    | Medium   | Function could be marked as external.                 |
|  393 | (SWC-120) Weak Sources of Randomness from Chain Attributes | Low | Potential use of "block.number" as source of randonmness. |
|  396 | (SWC-000) Unknown                                    | Medium   | Function could be marked as external.                 |
|  397 | (SWC-120) Weak Sources of Randomness from Chain Attributes | Low | Potential use of "blockhash" as source of randonmness.    |
|  397 | (SWC-120) Weak Sources of Randomness from Chain Attributes | Low | Potential use of "block.number" as source of randonmness. |
|  400 | (SWC-000) Unknown                                    | Medium   | Function could be marked as external.                 |
|  404 | (SWC-000) Unknown                                    | Medium   | Function could be marked as external.                 |
|  408 | (SWC-000) Unknown                                    | Medium   | Function could be marked as external.                 |
|  413 | (SWC-000) Unknown                                    | Medium   | Function could be marked as external.                 |
|  427 | (SWC-000) Unknown                                    | Medium   | Function could be marked as external.                 |
|  463 | (SWC-000) Unknown                                    | Medium   | Function could be marked as external.                 |
|  471 | (SWC-120) Weak Sources of Randomness from Chain Attributes | Low | Potential use of "block.number" as source of randonmness. |
|  476 | (SWC-120) Weak Sources of Randomness from Chain Attributes | Low | Potential use of "block.number" as source of randonmness. |
|  506 | (SWC-120) Weak Sources of Randomness from Chain Attributes | Low | Potential use of "block.number" as source of randonmness. |
|  525 | (SWC-000) Unknown                                    | Medium   | Function could be marked as external.                 |
|  532 | (SWC-120) Weak Sources of Randomness from Chain Attributes | Low | Potential use of "block.number" as source of randonmness. |
|  538 | (SWC-120) Weak Sources of Randomness from Chain Attributes | Low | Potential use of "block.number" as source of randonmness. |
|  546 | (SWC-000) Unknown                                    | Medium   | Function could be marked as external.                 |
|  550 | (SWC-000) Unknown                                    | Medium   | Function could be marked as external.                 |
|  554 | (SWC-000) Unknown                                    | Medium   | Function could be marked as external.                 |
```

Figure 1: MythX Results

THANK YOU FOR CHOOSING

// HALBORN