LUNARAY
BLOCKCHAINSECURITY

SMART CONTRACT SECURITY AUDIT REPORT

For Pandora

29 March 2022

lunaray.co

Table of Contents

1. Overview

On Mar 29, 2022, the security team of Lunaray Technology received the security audit request of the **PANDORA project**. The team completed the audit of the **PANDORA smart contract** on Mar 25, 2022. During the audit process, the security audit experts of Lunaray Technology and the PANDORA project interface Personnel communicate and maintain symmetry of information, conduct security audits under controllable operational risks, and avoid risks to project generation and operations during the testing process.

Through communicat and feedback with PANDORA project party, it is confirmed that the loopholes and risks found in the audit process have been repaired or within the acceptable range. The result of this PANDORA smart contract security audit: **Passed**

Audit Report Hash:

C704696AF35E0F610C34AEE9399A051E25C67BA5D035A879962F77637FAC4E81

2. Background

2.1 Project Description

Project name	PANDORA
Contract type	DeFi
Code language	Solidity
Public chain	PANDORA Chain DAO
Project address	http://pandoradao.net/
Contract file	PANDORA.sol

2.2 Audit Range

The project smart contract is in the PANDORA Chain DAO address：

Name	address
PANDORA.sol	0x102b22aDb425E65556685c1fc3470F379606d540

3. Project contract details

3.1 Contract Overview

PANDORA Contract

PANDORA contract has three main operational logic methods investment, openBox, updateDayBonus, these methods are used to invest, update rewards and other major contract logic, the contract exists two kinds of permissions, administrator privileges and ordinary user privileges, can call the above methods, in addition to the administrator can set certain calculation parameters.

3.2 Contract details

PANDORA Contract

Name	Parameter	Attributes
setDayImplement	none	internal
getEmpter	address x1 address x2	public
investment	address addr uint256 code	public
setDet	address a address b	internal
setPcdPrice	uint _price	onlyOwner
openBox	none	public
getUps	address up	public
mnuToShib	uint b	public
setNode	address _node	onlyOwner
updateDayBonus	none	public
DirectPushCompetition	none	internal
_bouns	address addr	internal
AutomaticRanking	address addr	internal
ranking	address addr uint z	internal
_DirectPush	address to uint256 _value	internal
setApi	none	public
_setAddress	address to uint256 _value	internal
getUserTermOfValidity	address addr	public
getReturnCommission	uint b	public
getReturnCommission25	uint b	public

getReturnCommission1	uint b	public
getReturnCommission2	uint b	public
random	uint256 randomyType	public
random1	uint256 randomyType	public
getLogs	uint logid	public
getContract	none	public
getRanking	none	public
shib	uint _value	onlyOwner
getUser	address addr	public

4. Audit details

4.1 Findings Summary

Severity	Found	Resolved	Acknowledged
● High	0	0	0
● Medium	0	0	0
● Low	3	0	3
● Info	2	0	2

4.2 Risk distribution

Name	Risk level	Repair status
onlyOwner permission	Low	Acknowledged
Possible information leakage	Info	Acknowledged
Redundancy Method	Info	Acknowledged
Random number of manageable risks	Low	Acknowledged
Possible integer overflow	Low	Acknowledged
Variables are updated	No	normal
Floating Point and Numeric Precision	No	normal
Default visibility	No	normal
tx.origin authentication	No	normal
Faulty constructor	No	normal
Unverified return value	No	normal
Insecure random numbers	No	normal
Timestamp Dependent	No	normal
Transaction order dependency	No	normal
Delegatecall	No	normal
Call	No	normal
Denial of Service	No	normal
Logical Design Flaw	No	normal
Fake recharge vulnerability	No	normal
Short address attack Vulnerability	No	normal
Uninitialized storage pointer	No	normal

Frozen account bypass	No	normal
Uninitialized	No	normal
Reentry attack	No	normal

4.3 Risk audit details

4.3.1. onlyOwner permission

- **Risk description**

PANDORA contract, setPcdPrice, setNode, shib method, onlyOwner can set part of the calculation parameters, there is a transfer function, if the private key is lost by malicious people control, or can lead to abnormal capital flows and shake the stability of the market.

```
function setPcdPrice(uint _price)public onlyOwner{
      daoBig50000=_price;
}

function setNode(address _node)public onlyOwner{
    nodes.push(_node);
}
```

- **Safety advice**

It is recommended to set TimeLock time lock for time constraint on administrator operation, and it is recommended to store this administrator key securely.

- **Repair Status**

PANDORA has confirmed the risk.

4.3.2. Possible information leakage

- **Risk description**

PANDORA contract, getUser method can get all the information of the user in the contract, any user can find other people's address information, currently can not completely rule out whether the address information will cause harm to the user and the project.

```
    function getUser(address addr)public view returns(address A,address
 B,uint C,uint D,uint E,uint F,uint G,uint H,uint I,uint J,uint K,uint
L){
        user storage _user=users[addr];
        A=_user.upAddress;
        B=_user.upAddress25;
        C=_user.level;
        D=_user.Number;
        E=_user.InvitationCode;
        F=0;
        G=_user.AllPeople;
        H=_user.DirectPushTime;
        I=_user.dayT;
        J=_user.Box;
        K=_user.okUp;
        L=_user.okUpsum;
    }
```

- **Safety advice**

It is recommended that the addr address in the getUser method be taken as msg.sender.

- **Repair Status**

PANDORA has confirmed the risk.

4.3.3. Redundancy Method

- **Risk description**

PANDORA contract, getUserTermOfValidity method returns null after entering address parameters, the contract does not have the logic to call this method, so the logic of getUserTermOfValidity method here is not clear.

```
    function getUserTermOfValidity(address addr)public view returns(uint){
        return 0;
    }
```

- **Safety advice**

It is recommended to clarify the logic of getUserTermOfValidity method, if it does not work, it is recommended to delete.

- **Repair Status**

PANDORA has confirmed the risk.

4.3.4. Random number of manageable risks

- **Risk description**

PANDORA contract, random and random1 methods both calculate random numbers through block.difficulty,block.timestamp, where randomyType variable exists in random number calculation, by tracking the randomyType variable is generated by block.timestamp and partially The miner may calculate the final result by the partially knowable variable when the value brings significantly larger gain.

```
    function setDet(address a,address b)internal{
        user storage _user=users[a];
        user storage _fall=users[b];
           _user.DirectPushTime=DailyDonus[7];
           _fall.Number++;
           if(_fall.okUp<25){
           _fall.okUp=getUps(b);
           }
            _user.okUp=5;
           _user.upAddress=b;
           InvitationCodeID++;
           _user.InvitationCode=InvitationCodeID;
           codeAddress[InvitationCodeID]=a;
      _DirectPush(b,investmentPrice);
      _setAddress(a,investmentPrice);
      address all=users[a].upAddress25;
      DailyDonus[1]+=DailyDonus[4];
      DailyDonus[2]+=DailyDonus[5];
      DailyDonus[3]+=DailyDonus[6];
      DailyDonus[10]+=DailyDonus[9];
      DailyDonus[11]++;
      if(dayAddress10.length<11){
          dayAddress10.push(a);
          uint mnu=random(block.timestamp+mnus);
          mnus++;
          if(dayAddress10.length==9){
             uint mnu1=random1(block.timestamp+mnus+1);
             users[dayAddress10[mnu1]].Box=mnuToShib(mnu);
            delete dayAddress10;
          }
      }

    function random(uint256 randomyType) public view returns(uint) {
        uint256 random = uint256(keccak256(abi.encodePacked(block.diffi
culty, block.timestamp,randomyType)));
        uint AAA=random%99;
        if(AAA>99){
            AAA=99;
        }
    return AAA;
    }

    function random1(uint256 randomyType) public view returns(uint) {
        uint256 random = uint256(keccak256(abi.encodePacked(block.diffi
culty, block.timestamp,randomyType)));
        uint AAA=random%9;
        if(AAA>9){
            AAA=9;
        }
    return AAA;
    }
```

- **Safety advice**

Suggest adding random number of uncontrollable parameters and random number of variables to reduce the impact on the contract logic.

- **Repair Status**

PANDORA has confirmed the risk.

4.3.5. Possible integer overflow

- **Risk description**

PANDORA contract, DirectPushCompetition, updateDayBonus multiple arithmetic operations using SafeMath safety function for arithmetic, through the SafeMath safety function to determine, found that the safety function can not completely avoid the risk of overflow, the same, for the main contract logic may also occur the risk of overflow.

```
library SafeMath {
    function add(uint256 a, uint256 b) internal pure returns (uint256)
{return a + b;}
    function sub(uint256 a, uint256 b) internal pure returns (uint256)
{return a - b;}
    function mul(uint256 a, uint256 b) internal pure returns (uint256)
{return a * b;}
    function div(uint256 a, uint256 b) internal pure returns (uint256)
{return a / b;}
    function mod(uint256 a, uint256 b) internal pure returns (uint256)
{return a % b;}

    function DirectPushCompetition()internal{
           if(codeAddress[1]!=address(0)){
               token.transfer(codeAddress[1],DailyDonus[10].mul(40).div
(100));
               logID++;
               LogMsg storage _log=UserLog[logID];
                     _log.token=codeAddress[1];
                     _log.log="Push straight to the first place";
                     _log.value=DailyDonus[10].mul(40).div(100);
                    _log.times=block.timestamp;
           }
           if(codeAddress[2]!=address(0)){
               token.transfer(codeAddress[2],DailyDonus[10].mul(275).di
v(1000));
               logID++;
               LogMsg storage _log=UserLog[logID];
                     _log.token=codeAddress[2];
                     _log.log="Push straight to the second place";
                     _log.value=DailyDonus[10].mul(275).div(1000);
                    _log.times=block.timestamp;
           }
           if(codeAddress[3]!=address(0)){
               token.transfer(codeAddress[3],DailyDonus[10].mul(175).di
v(1000));
               logID++;
               LogMsg storage _log=UserLog[logID];
                     _log.token=codeAddress[3];
                     _log.log="Straight to third place";
                     _log.value=DailyDonus[10].mul(175).div(1000);
                    _log.times=block.timestamp;
           }
           if(codeAddress[4]!=address(0)){
               token.transfer(codeAddress[4],DailyDonus[10].mul(10).div
(100));
               logID++;
               LogMsg storage _log=UserLog[logID];
                     _log.token=codeAddress[4];
                     _log.log="Straight to the fourth place";
                     _log.value=DailyDonus[10].mul(10).div(100);
                    _log.times=block.timestamp;
           }
           if(codeAddress[5]!=address(0)){
               token.transfer(codeAddress[5],DailyDonus[10].mul(5).div
(100));
               logID++;
               LogMsg storage _log=UserLog[logID];
                     _log.token=codeAddress[5];
                     _log.log="Straight to fifth place";
                     _log.value=DailyDonus[10].mul(5).div(100);
                    _log.times=block.timestamp;
           }
           setDayImplement();
    }
```

- **Safety advice**

It is recommended that all variables that can be influenced externally enter the contract logic using SafeMath safe functions for arithmetic, and it is recommended that the library SafeMath be updated.

- **Repair Status**

PANDORA has confirmed the risk.

4.3.6 Variables are updated

- **Risk description**

When there is a contract logic to obtain rewards or transfer funds, the coder mistakenly updates the value of the variable that sends the funds, so that the user can use the value of the variable that is not updated to obtain funds, thus affecting the normal operation of the project.

- **Audit Results : Passed**

4.3.7 Floating Point and Numeric Precision

- **Risk Description**

In Solidity, the floating-point type is not supported, and the fixed-length floating-point type is not fully supported. The result of the division operation will be rounded off, and if there is a decimal number, the part after the decimal point will be discarded and only the integer part will be taken, for example, dividing 5 pass 2 directly will result in 2. If the result of the operation is less than 1 in the token operation, for example, 4.9 tokens will be approximately equal to 4, bringing a certain degree of The tokens are not only the tokens of the same size, but also the tokens of the same size. Due to the economic properties of tokens, the loss of precision is equivalent to the loss of assets, so this is a cumulative problem in tokens that are frequently traded.

- **Audit Results : Passed**

4.3.8 Default Visibility

- **Risk description**

In Solidity, the visibility of contract functions is public pass default. therefore, functions that do not specify any visibility can be called externally pass the user. This can lead to serious vulnerabilities when developers incorrectly ignore visibility specifiers for functions that should be private, or visibility specifiers that can only be called from within the contract itself. One of the first hacks on Parity's multi-signature wallet was the failure to set the visibility of a function, which defaults to public, leading to the theft of a large amount of money.

- **Audit Results : Passed**

4.3.9 tx.origin authentication

- **Risk Description**

tx.origin is a global variable in Solidity that traverses the entire call stack and returns the address of the account that originally sent the call (or transaction). Using this variable for authentication in a smart contract can make the contract vulnerable to phishing-like attacks.

- **Audit Results : Passed**

4.3.10 Faulty constructor

- **Risk description**

Prior to version 0.4.22 in solidity smart contracts, all contracts and constructors had the same name. When writing a contract, if the constructor name and the contract name are not the same, the contract will add a default constructor and the constructor you set up will be treated as a normal function, resulting in your original contract settings not being executed as expected, which can lead to terrible consequences, especially if the constructor is performing a privileged operation.

- **Audit Results : Passed**

4.3.11 Unverified return value

- **Risk description**

Three methods exist in Solidity for sending tokens to an address: transfer(), send(), call.value(). The difference between them is that the transfer function throws an exception throw when sending fails, rolls back the transaction state, and costs 2300gas; the send function returns false when sending fails and costs 2300gas; the call.value method returns false when sending fails and costs all gas to call, which will lead to the risk of reentrant attacks. If the send or call.value method is used in the contract code to send tokens without checking the return value of the method, if an error occurs, the contract will continue to execute the code later, which will lead to the thought result.

- **Audit Results : Passed**

4.3.12 Insecure random numbers

- **Risk Description**

All transactions on the blockchain are deterministic state transition operations with no uncertainty, which ultimately means that there is no source of entropy or randomness within the blockchain ecosystem. Therefore, there is no random number function like rand() in Solidity. Many developers use future block variables such as block hashes, timestamps, block highs and lows or Gas caps to generate random numbers. These quantities are controlled pass the miners who mine them and are therefore not truly random, so using past or present block variables to generate random numbers could lead to a destructive vulnerability.

- **Audit Results : Passed**

4.3.13 Timestamp Dependency

- **Risk description**

In blockchains, data block timestamps (block.timestamp) are used in a variety of applications, such as functions for random numbers, locking funds for a period of time, and conditional statements for various time-related state changes. Miners have the ability to adjust the timestamp as needed, for example block.timestamp or the alias now can be manipulated pass the miner. This can lead to serious vulnerabilities if the wrong block timestamp is used in a smart contract. This may not be necessary if the contract is not particularly concerned with miner manipulation of block timestamps, but care should be taken when developing the contract.

- **Audit Results : Passed**

4.3.14 Transaction order dependency

- **Risk description**

In a blockchain, the miner chooses which transactions from that pool will be included in the block, which is usually determined pass the gasPrice transaction, and the miner will choose the transaction with the highest transaction fee to pack into the block. Since the information about the transactions in the block is publicly available, an attacker can watch the transaction pool for transactions that may contain problematic solutions, modify or revoke the attacker’s privileges or change the state of the contract to the attacker’s detriment. The attacker can then take data from this transaction and create a higher-level transaction gasPrice and include its transactions in a block before the original, which will preempt the original transaction solution.

- **Audit Results : Passed**

4.3.15 Delegatecall

- **Risk Description**

In Solidity, the delegatecall function is the standard message call method, but the code in the target address runs in the context of the calling contract, i.e., keeping msg.sender and msg.value unchanged. This feature supports implementation libraries, where developers can create reusable code for future contracts. The code in the library itself can be secure and bug-free, but when run in another application’s environment, new vulnerabilities may arise, so using the delegatecall function may lead to unexpected code execution.

- **Audit Results : Passed**

4.3.16 Call

- **Risk Description**

The call function is similar to the delegatecall function in that it is an underlying function provided pass Solidity, a smart contract writing language, to interact with external contracts or libraries, but when the call function method is used to handle an external Standard Message Call to a contract, the code runs in the environment of the external contract/function The call function is used to interact with an external contract or library. The use of such functions requires a determination of the security of the call parameters, and caution is recommended. An attacker could easily borrow the identity of the current contract to perform other malicious operations, leading to serious vulnerabilities.

- **Audit Results : Passed**

4.3.17 Denial of Service

- **Risk Description**

Denial of service attacks have a broad category of causes and are designed to keep the user from making the contract work properly for a period of time or permanently in certain situations, including malicious behavior while acting as the recipient of a transaction, artificially increasing the gas required to compute a function causing gas exhaustion (such as controlling the size of variables in a for loop), misuse of access control to access the private component of the contract, in which the Owners with privileges are modified, progress state based on external calls, use of obfuscation and oversight, etc. can lead to denial of service attacks.

- **Audit Results : Passed**

4.3.18 Logic Design Flaw

- **Risk Description**

In smart contracts, developers design special features for their contracts intended to stabilize the market value of tokens or the life of the project and increase the highlight of the project, however, the more complex the system, the more likely it is to have the possibility of errors. It is in these logic and functions that a minor mistake can lead to serious depasstions from the whole logic and expectations, leaving fatal hidden dangers, such as errors in logic judgment, functional implementation and design and so on.

- **Audit Results : Passed**

4.3.19 Fake recharge vulnerability

- **Risk Description**

The success or failure (true or false) status of a token transaction depends on whether an exception is thrown during the execution of the transaction (e.g., using mechanisms such as require/assert/revert/throw). When a user calls the transfer function of a token contract to transfer funds, if the transfer function runs normally without throwing an exception, the transaction will be successful or not, and the status of the transaction will be true. When balances[msg.sender] < _value goes to the else logic and returns false, no exception is thrown, but the transaction acknowledgement is successful, then we believe that a mild if/else judgment is an undisciplined way of coding in sensitive function scenarios like transfer, which will lead to Fake top-up vulnerability in centralized exchanges, centralized wallets, and token contracts.

- **Audit Results : Passed**

4.3.20 Short Address Attack Vulnerability

- **Risk Description**

In Solidity smart contracts, when passing parameters to a smart contract, the parameters are encoded according to the ABI specification. the EVM runs the attacker to send encoded parameters that are shorter than the expected parameter length. For example, when transferring money on an exchange or wallet, you need to send the transfer address address and the transfer amount value. The attacker could send a 19-passte address instead of the standard 20-passte address, in which case the EVM would fill in the 0 at the end of the encoded parameter to make up the expected length, which would result in an overflow of the final transfer amount parameter value, thus changing the original transfer amount.

- **Audit Results : Passed**

4.3.21 Uninitialized storage pointer

- **Risk description**

EVM uses both storage and memory to store variables. Local variables within functions are stored in storage or memory pass default, depending on their type. uninitialized local storage variables could point to other unexpected storage variables in the contract, leading to intentional or unintentional vulnerabilities.

- **Audit Results : Passed**

4.3.22 Frozen Account bypass

- **Risk Description**

In the transfer operation code in the contract, detect the risk that the logical functionality to check the freeze status of the transfer account exists in the contract code and can be passpassed if the transfer account has been frozen.

- **Audit Results : Passed**

4.3.23 Uninitialized

- **Risk description**

The initialize function in the contract can be called pass another attacker before the owner, thus initializing the administrator address.

- **Audit Results : Passed**

4.3.24 Reentry Attack

- **Risk Description**

An attacker constructs a contract containing malicious code at an external address in the Fallback function When the contract sends tokens to this address, it will call the malicious code. The call.value() function in Solidity will consume all the gas he receives when it is used to send tokens, so a re-entry attack will occur when the call to the call.value() function to send tokens occurs before the actual reduction of the sender’s account balance. The re-entry vulnerability led to the famous The DAO attack.

- **Audit Results : Passed**

5. Security Audit Tool

Tool name	Tool Features
Oyente	Can be used to detect common bugs in smart contracts
securify	Common types of smart contracts that can be verified
MAIAN	Multiple smart contract vulnerabilities can be found and classified
Lunaray Toolkit	self-developed toolkit

Disclaimer：

Lunaray Technology only issues a report and assumes corresponding responsibilities for the facts that occurred or existed before the issuance of this report, Since the facts that occurred after the issuance of the report cannot determine the security status of the smart contract, it is not responsible for this.

Lunaray Technology conducts security audits on the security audit items in the project agreement, and is not responsible for the project background and other circumstances, The subsequent on-chain deployment and operation methods of the project party are beyond the scope of this audit.

This report only conducts a security audit based on the information provided by the information provider to Lunaray at the time the report is issued, If the information of this project is concealed or the situation reflected is inconsistent with the actual situation, Lunaray Technology shall not be liable for any losses and adverse effects caused thereby.

There are risks in the market, and investment needs to be cautious. This report only conducts security audits and results announcements on smart contract codes, and does not make investment recommendations and basis.

LUNARAY

BLOCKCHAIN SECURITY

https://lunaray.co

https://github.com/lunaraySec

https://twitter.com/lunaray_Sec

http://t.me/lunaraySec