LUNARAY
BLOCKCHAINSECURITY

Smart Contract Security Audit Report

For **Wilderness Survival**

24 November 2021

https://lunaray.co

Table of Contents

1. Overview

On Nov 15, 2021, the security team of Lunaray Technology received the security audit request of the **Wilderness Survival project**. The team completed the audit of the **Wilderness Survival smart contract** on Nov 24, 2021. During the audit process, the security audit experts of Lunaray Technology and the Wilderness Survival 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 Wilderness Survival 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 Wilderness Survival smart contract security audit: **passed**

Audit Report MD5：F872DF66D36CC5A1AFE487EC5B8786B3

2. Background

2.1 Project Description

Project name	Wilderness Survival
Contract type	NFT, GameFi
Code language	Solidity
Public chain	Binance Smart Chain
Project address	https://wsurvival.io/
Contract file	Meat.sol, Iron.sol, Crystal.sol, Equipment.sol, LootboxNFT.sol, NftMarket.sol
Project Description	The Wilderness Survival chain game is developed by the original GreenHell team. It continues the previous setting and combines Metaverse + nft to create a new immersive survival chain game, an open, interactive, and decentralized virtual world based on the bsc blockchain.

2.2 Audit Range

Wilderness Survival officially provides smart contract files and corresponding MD5：

Name	Address
Crystal.sol	DDC041251C54D2F885D636EB88DB9AFB
iron.sol	4BBD9E6E524F4CA8E9A5F9E55B7A1100
meat.sol	E9DE7C857040EA7F065806B5EC7ED2BB
Equipment.sol	68A947D8B4E82452F63FA81B662FC886
LootboxNFT.sol	8FF34A7BC0E92D2B6196BA53F8FB2647
NFTMarket.sol	59C4890B6EE342DABD2A86CD82CED5B3

2.3 Findings Summary

Severity	Found	Resolved	Acknowledged
● High	0	0	0
● Medium	1	0	1
● Low	4	0	4
● Info	2	0	2

3. Project Contract Details

3.1 Directory Structure

```
└─Wilderness Survival
    ├──Material
    |      Crystal.sol
    |      Iron.sol
    |      Meat.sol
    └──NFT
         Equipment.sol
         LootboxNFT.sol
         NFTMarket.sol
```

3.2 Contract Details

LootboxNFT

Name	Parameter	Attributes
LootboxNFT	none	public
updateContract	address equipmentContract address crystalContract address ironContract address meatContract	onlyOwner
updateMarketContract	address newMarket	onlyOwner
updateSellExpect	uint256 type0Amount uint256 type1Amount	onlyOwner
updateMaterialReward	uint256 newReardAmount	onlyOwner
getSaleAmount	none	external
buyOfficial	address to uint256 tokenType	onlyMarket
safeMint	address to uint256 tokenType	onlyOwner
openLootbox	uint256 tokenId	onlyPerson
issueMaterial	address to	private
issueEquipment	address to	private
randomNumber	none	private
updateTokenURI	uint256 tokenTypeId string _tokenURI	onlyOwner
updateBaseURI	string baseURI_	onlyOwner

NftMarket

Name	Parameter	Attributes
NftMarket	none	external
enableSales	bool _salesEnabled	onlyOwner
setFeeReceiver	addresspayable _walletAddress	onlyOwner
setFeeRate	uint256 _rate	onlyOwner
setLootbox	address lootbox_	onlyOwner
setLootboxPrice	uint256 _price	onlyOwner
getSalesList	none	external
getSaleIndexesOf	address account	public
getSaleInfo	uint256 id	external
onERC721Received	address operator address from uint256 tokenId bytes data	public
sell	uint256 _tokenId uint256 _price address nftAddress	public
cancelSell	uint _saleIndex	public
changeIndex	uint _saleIndex	private
buy	uint _saleIndex uint256 tokenId	public
buyOfficial	uint256 lootboxType	public
emergencyBNBWithdraw	none	onlyOwner

Iron

Name	Parameter	Attributes
Iron	none	public
updateLootbox	address newLootbox	onlyOwner
name	none	public
symbol	none	public
decimals	none	public
totalSupply	none	public
balanceOf	address account	public
transfer	address recipient uint256 amount	public
allowance	address owner address spender	public
transferFrom	address sender address recipient uint256 amount	public
increaseAllowance	address spender uint256 addedValue	public
decreaseAllowance	address spender uint256 subtractedValue	public
_transfer	address sender address recipient uint256 amount	internal
_mint	address account uint256 amount	internal
_burn	address account uint256 amount	internal
_approve	address owner address spender uint256 amount	internal
_beforeTokenTransfer	address from address to uint256 amount	internal
_afterTokenTransfer	address from address to uint256 amount	internal
emergencyTokenWithdraw	uint256 _amount	onlyOwner

Meat

Name	Parameter	Attributes
Meat	none	public
updateLootbox	address newLootbox	onlyOwner
name	none	public
symbol	none	public
decimals	none	public
totalSupply	none	public
balanceOf	address account	public
transfer	address recipient uint256 amount	public
allowance	address owner address spender	public
transferFrom	address sender address recipient uint256 amount	public
increaseAllowance	address spender uint256 addedValue	public
decreaseAllowance	address spender uint256 subtractedValue	public
_transfer	address sender address recipient uint256 amount	internal
_mint	address account uint256 amount	internal
_burn	address account uint256 amount	internal
_approve	address owner address spender uint256 amount	internal
_beforeTokenTransfer	address from address to uint256 amount	internal
_afterTokenTransfer	address from address to uint256 amount	internal
emergencyTokenWithdraw	uint256 _amount	onlyOwner

Crystal

Name	Parameter	Attributes
Crystal	none	public
updateLootbox	address newLootbox	onlyOwner
name	none	public
symbol	none	public
decimals	none	public
totalSupply	none	public
balanceOf	address account	public
transfer	address recipient uint256 amount	public
allowance	address owner address spender	public
transferFrom	address sender address recipient uint256 amount	public
increaseAllowance	address spender uint256 addedValue	public
decreaseAllowance	address spender uint256 subtractedValue	public
_transfer	address sender address recipient uint256 amount	internal
_mint	address account uint256 amount	internal
_burn	address account uint256 amount	internal
_approve	address owner address spender uint256 amount	internal
_beforeTokenTransfer	address from address to uint256 amount	internal
_afterTokenTransfer	address from address to uint256 amount	internal
emergencyTokenWithdraw	uint256 _amount	onlyOwner

EquipmentNFT

Name	Parameter	Attributes
EquipmentNFT	none	public
initEquipment	none	internal
updateEquipment	uint256 tokenType uint256 outputInterval uint256 outputMin uint256 outputMax uint256 costCrystal uint256 costMeat uint256 durability uint256 crystalForCraft uint256 ironForCraft	onlyOwner
queryEquipmentConfigInfosByType	uint256 typeId	public
setContractAddr	address _lootboxAddr address _wildernessSurvivalAddr	onlyOwner
updateTokenURI	uint256 tokenType string _tokenURI	onlyOwner
updateBaseURI	string baseURI_	onlyOwner
_approve	address to uint256 tokenId	internal
tokensIdOf	address account	public
getEquipmentConfigList	none	external
uniqueTokenInfo	uint256 tokenId	public
balanceOf	address owner	public
typeTotalSupply	uint256 tokenType	public
ownerOf	uint256 tokenId	public

Name	Parameter	Attributes
name	none	public
symbol	none	public
tokenURI	uint256 tokenId	public
baseURI	none	public
tokenOfOwnerByIndex	address owner uint256 tokenType uint256 index	public
totalSupply	none	public
approve	address to uint256 tokenId	public
getApproved	uint256 tokenId	public
setApprovalForAll	address operator bool approved	public
isApprovedForAll	address owner address operator	public
transferFrom	address from address to uint256 tokenId	public
safeTransferFrom	address from address to uint256 tokenId bytes _data	public
_safeTransfer	address from address to uint256 tokenId bytes _data	internal
_exists	uint256 tokenId	internal
_isApprovedOrOwner	address spender uint256 tokenId	internal
safeMint	address to uint256 tokenType bytes _data	onlyContract
updateEquipmentInfo ByDurability	uint256 tokenId uint256 durability	onlyContract
_mint	address to uint256 tokenType	internal

Name	Parameter	Attributes
	uint256 tokenId	
_burn	uint256 tokenId	internal
_transfer	address from address to uint256 tokenId	internal
_removeToken	address from uint256 tokenType uint256 tokenId	private
_addToken	address to uint256 tokenType uint256 tokenId	private
tokensInfoOf	address account	public
tokenDurability	uint256 _tokenId	public
_checkOnERC721Received	address from address to uint256 tokenId bytes _data	private
_beforeTokenTransfer	address from address to uint256 tokenId	internal

4. Audit Details

4.1 Risk Distribution

Name	Risk level	Status
Administrator Permissions	Low	Acknowledged
No events added	Info	Acknowledged
onlyPerson does not add parameters	Low	Acknowledged
Random number controllable	Medium	Acknowledged
Misinitialised storage pointers	Low	Acknowledged
Redundant codes	Info	Acknowledged
Determining incoming address parameters	Low	Acknowledged
Unused Local Variables	No	Passed
Gas Consumption Due to for Loops	No	Passed
Self-transfer Issues	No	Passed
Floating Point and Numeric Precision	No	Passed
Default Visibility	No	Passed
tx.origin Authentication	No	Passed
Wrong constructor	No	Passed
Unverified Return Value	No	Passed
Timestamp Dependent	No	Passed
Transaction order Dependence	No	Passed
Delegatecall	No	Passed

Call	No	Passed
Denial of Service	No	Passed
Logical Design Flaw	No	Passed
Fake Recharge Vulnerability	No	Passed
Short Address Attack	No	Passed
Uninitialized Storage Pointer	No	Passed
Frozen Account Bypass	No	Passed
Uninitialized	No	Passed
Integer Overflow	No	Passed

4.2 Risk Audit Details

4.2.1 Administrator Permissions

- **Risk description**

Meat, Iron, Crystal, EquipmentNFT, LootboxNFT, NftMarket contract, emergencyTokenWithdraw, updateContract method and many other methods can perform sensitive operations that could lead to abnormal funds if the administrator's private key is controlled by a malicious person. Loss of funds and destabilisation of the market, as shown in the following code:

```
    function emergencyTokenWithdraw(uint256 _amount) public onlyOwner {
        require(_amount <= balanceOf(address(this)), 'not enough token
');
        IERC20(address(this)).safeTransfer(address(msg.sender), _amoun
t);
    }

    function updateContract(address equipmentContract, address crystalC
ontract, address ironContract, address meatContract) public onlyOwner {
        wildernessSurvivalEquipmentContract = equipmentContract;
        wildernessSurvivalCrystalContract = crystalContract;
        wildernessSurvivalIronContract = ironContract;
        wildernessSurvivalMeatContract = meatContract;
    }

    function updateMarketContract(address newMarket) external onlyOwner
 {
        require(newMarket != address(0), "LootboxNFT: Market contract a
ddress can not be address zero");
        marketContract = newMarket;
    }
```

- **Safety advice**

It is recommended that a TimeLock be set to time-bind administrator actions; it is recommended that this administrator key be stored securely.

- **Repair status**

This risk is known to the WildernessSurvival Project.

4.2.2 No events added

- **Risk description**

Meat, Iron, Crystal, EquipmentNFT, LootboxNFT, NftMarket contract, emergencyTokenWithdraw, updateContract method and many other methods can perform sensitive operations, in order to keep users and administrators informed of contract operation details, it is recommended that Add event logging, as shown in the following code:

```
    function emergencyTokenWithdraw(uint256 _amount) public onlyOwner {
        require(_amount <= balanceOf(address(this)), 'not enough token');
        IERC20(address(this)).safeTransfer(address(msg.sender), _amount);
    }

    function updateContract(address equipmentContract, address crystalContract, address ironContract, address meatContract) public onlyOwner {
        wildernessSurvivalEquipmentContract = equipmentContract;
        wildernessSurvivalCrystalContract = crystalContract;
        wildernessSurvivalIronContract = ironContract;
        wildernessSurvivalMeatContract = meatContract;
    }

    function updateMarketContract(address newMarket) external onlyOwner {
        require(newMarket != address(0), "LootboxNFT: Market contract address can not be address zero");
        marketContract = newMarket;
    }
```

- **Safety advice**

It is recommended that event logging be added for sensitive operations.

- **Repair status**

This risk is known to the WildernessSurvival Project.

4.2.3 onlyPerson does not add parameters

- **Risk description**

LootboxNFT contract, onlyPerson modifier by input parameters to determine whether to continue to run, and the onlyPerson modifier used in the openLootbox method does not pass in the address parameter, in order to avoid the occurrence of contract exception execution, it is recommended to add the onlyPerson modifier parameter, as shown in the following code:

```
    function openLootbox(uint256 tokenId) onlyPerson external {

    modifier onlyPerson(address caller) {
        require(!caller.isContract, "LootboxNFT: caller is not a person
");
        _;
    }
```

- **Safety advice**

It is recommended to add the onlyPerson modifier parameter.

- **Repair status**

This risk is known to the WildernessSurvival Project.

4.2.4 Random number controllable

- **Risk description**

The LootboxNFT contract, the issueMaterial method and the issueEquipment method both call the randomNumber method to obtain a rand random number and use that random number to make subsequent conditional judgements, as shown in the following code:

```
    function issueMaterial(address to) private returns(string memory to
kenName) {
        uint256 rand = randomNumber();
        if(rand < 400) {
            IERC20(wildernessSurvivalIronContract).transferFrom(wildern
essSurvivalIronContract, to, materialReward);
            tokenName = "Iron";
        } else if(rand < 700) {
            IERC20(wildernessSurvivalCrystalContract).transferFrom(wild
ernessSurvivalCrystalContract, to, materialReward);
            tokenName = "Crystal";
        } else if(rand < 1000) {
            IERC20(wildernessSurvivalMeatContract).transferFrom(wildern
essSurvivalMeatContract, to, materialReward);
            tokenName = "Meat";
        }
    }
    function issueEquipment(address to) private returns(string memory t
okenName) {
        uint256 rand = randomNumber();
        uint256 tokenTypeId;
        if(rand < 480) {
            tokenTypeId = 5;
            tokenName = "Animal net";
        } else if(rand < 800) {
            tokenTypeId = 2;
            tokenName = "Hammer";
        } else if(rand < 900) {
            tokenTypeId = 8;
            tokenName = "Loot gathering puppy";
        } else if(rand < 948) {
            tokenTypeId = 6;
            tokenName = "Bow arrow";
        } else if(rand < 980) {
            tokenTypeId = 3;
            tokenName = "Forging hammer";
        } else if(rand < 988) {
            tokenTypeId = 7;
            tokenName = "Shotgun";
        } else if(rand < 995) {
            tokenTypeId = 4;
            tokenName = "Smelting machine";
        } else if(rand < 1000) {
            tokenTypeId = 1;
            tokenName = "Mining machine";
        }
        IEquipment(wildernessSurvivalEquipmentContract).safeMint(to, to
kenTypeId, "");
    }
```

Continue following the randomNumber method as follows:

```
    function randomNumber() private returns(uint256) {
        uint256 rand =
uint256(keccak256(abi.encodePacked(block.timestamp, gasleft(),
randNonce))) % 1000;
        randNonce = rand;
        return rand;
    }
```

The randomNumber method shows that the random number rand is generated by three main parameters, block.timestamp, gasleft(), and randNonce.

block.timestamp: the actual stamp of the current block, in seconds, this value can be controlled

gasleft(): the gas value of the transaction minus the gas of the transaction execution until now, this value can be calculated by executing the method once or several times after the execution of the required gas to the current position, and finally and the input gas value subtracted to obtain the remaining gas, through the analysis of the value can be controlled.

randNonce: randNonce is the random number rand of the last call, which can be executed twice in succession to obtain the first definite value, thus bringing the known rand into the second calculation, thus making randNonce.

Ultimately, since all three parameters are controllable, the attacker may be able to obtain equipment or materials of higher value by detailed calculations.

- **Safety advice**

It is recommended that random number security be added.

- **Repair status**

This risk is known to the WildernessSurvival Project.

4.2.5 Misinitialised storage pointers

(1) - **Risk description**

NftMarket contract, buy method, Equipement contract, _removeToken method does not determine if the passed in parameters _saleIndex, from, tokenType exist as an index, EVM uses both storage and memory to store variables, local variables within functions are stored in storage or memory by default depending on In the way Solidity works, state variables are stored in the contract's Slot in the order in which they appear in the contract, and incorrectly initialized local storage variables can point to other unexpected storage variables in the contract, leading to intentional or unintentional vulnerabilities, as shown in the following code:

```
    /// @notice buy nft
    /// @param _saleIndex nft unique ID
    function buy(uint _saleIndex, uint256 tokenId) public nonReentrant 
payable {
        require(salesEnabled, "sales are closed");
        SalesObject storage obj = salesObjects[_saleIndex];
        require(obj.startTime <= now, "not yet for sale");
        require(obj.tokenId == tokenId, "wrong token, reload page");
        require(msg.sender != obj.seller, "cant buy from yourself");

        uint256 price = obj.price;
        uint256 tipsFee = price.mul(feeRate).div(1000);
        uint256 purchase = price.sub(tipsFee);

        require (msg.value >= price, "your price is too low");
        if (tipsFee > 0){
            feeReceiver.transfer(tipsFee);
        }
        obj.seller.transfer(purchase);

        IERC721 nftContract = IERC721(obj.nft);
        nftContract.safeTransferFrom(address(this), msg.sender, obj.tok
enId);

        changeIndex(_saleIndex);
        emit Buy(obj.id, obj.tokenId, msg.sender, price, tipsFee);
    }

    function _removeToken(address from, uint256 tokenType, uint256 toke
nId) private {
        uint256[] storage tokens = _holderTokens[from][tokenType];
        for(uint256 i = 0; i < tokens.length; i++){
            if(tokens[i] == tokenId){
                tokens[i] = tokens[tokens.length - 1];
                tokens.pop();
                break;
            }
        }
    }
```

- **Safety advice**

It is recommended that the passed parameter _saleIndex be checked to determine if it exists as an index.

- **Repair status**

This risk is known to the WildernessSurvival Project.

(2) - **Risk description**

Equipement contract, tokenOfOwnerByIndex method, which is a view method and the object tokenIds initialised with storage has not changed state, increasing the risk of memory being overwritten and gas consuming too many transactions being rolled back, as shown in the following code.

```
        function tokenOfOwnerByIndex(address owner, uint256 tokenType,
uint256 index) public view returns (uint256) {
        uint256[] storage tokenIds =  _holderTokens[owner][tokenType];
        require(index < tokenIds.length, "EquipmentNFT: index must be l
ess than the array length");
        return tokenIds[index];
    }
```

- **Safety advice**

It is recommended to use memory instead of storage to initialise objects.

- **Repair status**

This risk is known to the WildernessSurvival Project.

4.2.6 Redundant codes

- **Risk description**

Crystal contract, the onlyContract modifier is not used and is redundant code, as shown in the following code:

```
    modifier onlyContract() {
        require(lootbox == msg.sender, "ERC20: only contract can call this function");
        _;
    }
```

- **Safety advice**

It is recommended that redundant code be removed.

- **Repair status**

This risk is known to the WildernessSurvival Project.

4.2.7 Determining incoming address parameters

- **Risk description**

Equipment contract, _transfer method, does not determine whether the incoming address from, to is the same address, and the method's transferFrom, _safeTransfer, safeTransferFrom are also not determined, as shown in the following code:

```
    function _transfer(address from, address to, uint256 tokenId) inter
nal virtual {
        require(ownerOf(tokenId) == from, "ERC721: transfer of token th
at is not own");
        require(to != address(0), "ERC721: transfer to the zero address
");

        _beforeTokenTransfer(from, to, tokenId);

        // Clear approvals from the previous owner
        _approve(address(0), tokenId);

        // Remove token id from 'from'
        uint256 tokenType =  tokenEquipments[tokenId].tokenType;
        _removeToken(from, tokenType, tokenId);
        _holderSupply[from] = _holderSupply[from].sub(1);

        // Add token id to 'to'
        _addToken(to, tokenType, tokenId);
        _holderSupply[to] = _holderSupply[to].add(1);

        // Update token owner
        _owners[tokenId] = to;

        emit Transfer(from, to, tokenId);
    }
```

- **Safety advice**

It is recommended that the two addresses passed in at the beginning of this internal method are not determined to be the same address.

- **Repair status**

This risk is known to the WildernessSurvival Project.

4.2.8. Unused local variables

- **Risk Description**

In Solidity contracts, there may be operations that assign values to partial variables in multiple methods due to coders' negligence, but there is no operation on the variable in the logic afterwards. If this issue occurs, it may increase the gas consumption for contract deployment and operation.

- **Audit Results : Passed**

4.2.9. Gas Consumption Due to for Loops

- **Risk Description**

In Solidity contract, due to some logic functions need, may use for loop for processing, if there is no strict judgment on for loop, may make the loop keep running, consume gas or affect the contract logic, if this problem occurs, may increase the contract running gas consumption, serious can cause the contract can not enter the normal logic.

- **Audit Results : Passed**

4.2.10 Self-transfer issues

- **Risk Description**

In Solidity contracts, there will be transfer function, most contracts use ERC20 transfer method to handle the transfer, but some of the transfer may require developers to write the transfer logic due to the need to fit the project logic function, if there are rewards or other means to obtain funds, self-transfer can be made to obtain a large number of fees normal funds acquisition.

- **Audit Results : Passed**

4.2.11 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 Findings : Passed**

4.2.12 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**

4.2.13 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**

4.2.14 Wrong 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**

4.2.15 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**

4.2.16 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**

4.2.17 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**

4.2.18 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**

4.2.19 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**

4.2.20 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**

4.2.21 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**

4.2.22 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**

4.2.23 Short Address Attack

- **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**

4.2.24 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 Findings : Passed**

4.2.25 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**

4.2.26 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**

4.2.27 Integer Overflow

- **Risk Description**

Integer overflows are generally classified as overflows and underflows. The types of integer overflows that occur in smart contracts include three types: multiplicative overflows, additive overflows, and subtractive overflows. In Solidity language, variables support integer types in steps of 8, from uint8 to uint256, and int8 to int256, integers specify fixed size data types and are unsigned, for example, a uint8 type , can only be stored in the range 0 to 2^8-1, that is, [0,255] numbers, a uint256 type can only store numbers in the range 0 to 2^256-1. This means that an integer variable can only have a certain range of numbers represented, and cannot exceed this formulated range. Exceeding the range of values expressed pass the variable type will result in an integer overflow vulnerability.

- **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

BLOCKCHAINSECURITY

https://lunaray.co

https://github.com/lunaraySec

https://twitter.com/lunaray_Sec

http://t.me/lunaraySec