LUNARAY
BLOCKCHAINSECURITY

Smart Contract Security audit Report

For Metaverse Player One

19 November 2021

https://lunaray.io

Table of Contents

1. Overview

On Nov 4, 2021, the security team of Lunaray Technology received the security audit request of the **Metaverse Player One project**. The team completed the audit of the **Metaverse Player One smart contract** on Nov 19, 2021. During the audit process, the security audit experts of Lunaray Technology and the Metaverse Player One 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 Metaverse Player One 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 Metaverse Player One smart contract security audit: **passed**

Audit Report MD5：0A0F6D6A112206B8C4C580BBA6DAAC20

2. Background

2.1 Project Description

Project name	Metaverse Player One
Contract type	NFT, DeFi, GameFi
Code language	Solidity
Public chain	OKExChain
Project address	http://www.ufox.io
Contract file	Card.sol, ContractOwner.sol, ERC165.sol, ERC721.sol, ERC721EX.sol, KKExchange.sol, Manager.sol, Member.sol, newCardMIne.sol, newMortgageBase.sol, Slot.sol, Token.sol
Project Description	Metaverse Player One perfectly integrates the revenue farming of games and DeFi, allowing players to get rich on-chain revenue while playing entertainment.

2.2 Audit Range

Metaverse Player One officially provides contract documents and documents corresponding to MD5：

Name	Hash
Card.sol	79C1A26853B161F1F5AA410CA44C72AD
ContractOwner.sol	AB9583BB5119F1C0790918A473DCA29F
ERC165.sol	6B4F3B7C32794ECDFA0DC4E7F919F68D
ERC721.sol	A6384BB3D75516A982D99AB4A8D2CDF4
ERC721EX.sol	D3B280E9F112E0E8C376291BCDE3CB9D
KKExchange.sol	DB35CBC55F9108AB2759A0F401AF2020
Manager.sol	5F9866FBB393C08423692F18FD90B8ED
Member.sol	F26ABDD2918D243A6888FE5AEBAB0389
newCardMIne.sol	0281F4FCEDAA6EEB6BF654B0C85C14ED
Slot.sol	D620F2DC417220A29EC000E43738CCE7
Token.sol	265D6FF9BF39B3AE2E066C481AB2FA2B
newMortgageBase.sol	81E435D8437FE5C178689D9E82BA6131

2.3 Findings Summary

Severity	Found	Resolved	Acknowledged
High	0	0	0
Medium	0	0	0
Low	4	4	0
Info	0	0	0

3. Project contract details

3.1 Directory Structure

└─Metaverse Player One

| Card.sol

| ContractOwner.sol

| ERC165.sol

| ERC721.sol

| ERC721EX.sol

| KKExchange.sol

| Manager.sol

| Member.sol

| newCardMIne.sol

| newMortgageBase.sol

| Slot.sol

| Token.sol

3.2 Contract details

MortgageBase

Name	Parameter	Attributes
calcInterest	address owner	public
deposit	address owner uint256 _amount	internal
_Deposit	address owner uint256 _amount	internal
updataPool	none	internal
getPrincipal	address owner uint256 amount	internal
pullAll	address owner	internal
stopMortgage	none	external

Slot

Name	Parameter	Attributes
setRarityExp	uint256 rarity uint256 exp	external
setLevelConfig	uint256 level uint256 exp int256 buffer	external
getUserInfo	address owner	external
getUserFight	address owner	external
getSlotInfo	address owner uint256 cardType	external
_onFightChanged	address owner	internal
removeCard	uint256 cardType	external
removeAllCards	none	external

CardMine

Name	Parameter	Attributes
updateFight	address owner int256 _fight	external
withdraw	bytes data	external

Card

Name	Parameter	Attributes
setRarotyValues	uint256 index uint256 price	public
setRarityFight	uint256 rarity int256 fight	external
setBurnLockDuration	uint256 duration	external
setPackage	address package bool enable	external
mint	address to uint256 cardIdPre	external
burn	uint256 cardId	external
withdraw	none	external
getFight	uint256 cardId	external
tokenURI	uint256 cardId	public

UFO

Name	Parameter	Attributes
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
approve	address spender uint256 amount	public
transferFrom	address sender address recipient uint256 amount	public
_transfer	address sender address recipient uint256 amount	internal
_approve	address owner address spender uint256 amount	internal

Member

Name	Parameter	Attributes
setManager	address addr	ContractOwnerOnly

KKExchange

Name	Parameter	Attributes
setSupportToken	address token bool isSupport	ContractOwnerOnly
createOrder	uint256 cardid uint256 tradeAmount address token	external
takeOrder	uint256 cardid	external
cancelOrder	uint256 cardid	external
withdraw	address token	external
calcWithdraw	address user address token	external
getMakerOrder	uint256 cardid	external
getTradeOrder	bytes32 orderid	external

CardMine

Name	Parameter	Attributes
updateFight	address owner int256 _fight	external
withdraw	bytes data	external

Manager

Name	Parameter	Attributes
setMember	string name address member	ContractOwnerOnly
setUserPermit	address user string permit bool enable	ContractOwnerOnly
getTimestamp	none	external

4. Audit details

4.1 Risk distribution

Name	Risk level	Status
Administrator Permissions	Low	Resolved
Variable update	Low	Resolved
Integer Overflow	Low	Resolved
Numerical accuracy	No	Passed
Default visibility	No	Passed
tx.origin authentication	Low	Resolved
Numerical accuracy	No	Passed
Default visibility	No	Passed
tx.origin authentication	No	Passed
Wrong constructor	No	Passed
Unverified return value	No	Passed
Insecure random number	No	Passed
Timestamp dependent	No	Passed
Transaction order dependence	No	Passed
Delegatecall	No	Passed
Call	No	Passed
Denial of service	No	Passed
Logical design flaws	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
Reentry attack	No	Passed
Integer Overflow	No	Passed

4.2 Risk audit details

4.2.1 Arithmetic operations not using safemath safety functions

- **Risk description**

In the Card contract, the burnForSlot method, withdraw method and many other arithmetic operations do not use safe functions, in order to avoid integer overflow problems, it is recommended that all arithmetic operations use SafeMath safe functions, the part of the code that does not use safe functions, as shown in the following code:

```
function burnForSlot(uint256[] memory cardIds) external {
    uint256 length = cardIds.length;
    address owner = msg.sender;
    uint256 tokenAmount = 0;

    for (uint256 i = 0; i != length; ++i) {
        uint256 cardId = cardIds[i];
        require(owner == _owners[cardId], "you are not owner");
        _burn(cardId);
        tokenAmount = rarityValues[uint16(cardId >> 192)];
    }
    LockedToken storage lt = upgradeLockedTokens[owner];
    uint256 _now = block.timestamp;
    if (_now < lt.lockTime + UPGRADE_LOCK_DURATION) {
        uint256 amount = lt.locked * (_now - lt.lockTime)
            / UPGRADE_LOCK_DURATION;
        lt.locked = lt.locked - amount + tokenAmount;
        lt.unlocked += int256(amount);
    } else {
        lt.unlocked += int256(lt.locked);
        lt.locked = tokenAmount;
```

```
    }
    lt.lockTime = _now;
    Slot(manager.members("slot")).upgrade(owner, cardIds);
  }

  function withdraw() external {
    LockedToken storage lt = upgradeLockedTokens[msg.sender];
    int256 available = lt.unlocked;
    uint256 _now = block.timestamp;
    if (_now < lt.lockTime + UPGRADE_LOCK_DURATION) {
      available += int256(lt.locked * (_now - lt.lockTime)
        / UPGRADE_LOCK_DURATION);
    } else {
      available += int256(lt.locked);
    }
    require(available > 0, "no token available");
    lt.unlocked -= available;

    // not check result to save gas
    IERC20(manager.members("token")).transfer(msg.sender, uint256(available));
  }
```

- **Safety advice**

It is recommended that all arithmetic operations use the SafeMath safety function.

- **Repair Status**

The risk has been officially modified through communication with the Metaverse Player One officials.

4.2.2 Failure to determine that sender and recipient are not the same address

- **Risk description**

UFO contracts, the _transfer method does not determine that the sender's address and the received address are not the same address. To avoid security problems, it is recommended that the sender's address and the received address cannot be the same address, and the _transfer method as shown in the following code:

```
function _transfer(address sender, address recipient, uint256 amount) internal virtual {
    require(sender != address(0), "ERC20: transfer from the zero address");
    require(recipient != address(0), "ERC20: transfer to the zero address");
    require(_balances[sender] >= amount, "ERC20: sender balance not enough");
    _balances[sender] = _balances[sender] - amount;
    _balances[recipient] = _balances[recipient] + amount;
    emit Transfer(sender, recipient, amount);
}
```

- **Safety advice**

It is recommended that the sender's address and the receiver's address should not be the same address

- **Repair Status**

The risk has been officially modified through communication with the Metaverse Player One officials.

4.2.3 No lastBlock < startBlock

- **Risk description**

MortgageBase contract, updataPool method, by determining whether the lastBlock < startBlock is not found, so whether the determination here is unnecessary as shown in the following code:

```
function updataPool() internal{
    uint256 balance = totalAmounts;
    if(balance == 0){
        poolShare = 0;
        lastBlock = block.number;
        return;
    }
    if(lastBlock < startBlock) {
        lastBlock = startBlock;
    }
    if(startBlock > block.number){
        poolShare = 0;
        return;
    }
    uint256 m;
    uint256 d;
    while(lastUpdataPhaseBlock < block.number){
        singleBlockreward = singleBlockreward.sub(singleBlockreward.mul(1).div(10));
        m = lastUpdataPhaseBlock.sub(lastBlock,'22222222222').mul(singleBlockreward);
        d = balance;
        lastBlock = lastUpdataPhaseBlock;
        lastUpdataPhaseBlock = lastUpdataPhaseBlock.add(phase);
        poolShare = poolShare.add(m.div(d));
    }
```

```
    m = block.number.sub(lastBlock,'22222222222').mul(singleBlockreward);
    d = balance;
    lastBlock = block.number;
    poolShare = poolShare.add(m.div(d));
  }
```

- **Safety advice**

If there is no case where lastBlock < startBlock, it is recommended to remove this judgement.

- **Repair Status**

The risk has been officially modified through communication with the Metaverse Player One officials.

4.2.4 Unused local variables

- **Risk description**

ERC721Ex contract, _burn method, there is an operation to assign a value to the owner variable, but there is no operation on the owner variable in the logic that shown in the following code:

```
function _burn(uint256 tokenId)  internal {
    address owner = _owners[tokenId];
    _burnOld(tokenId);


    if (_tokenApprovals[tokenId] != address(0)) {
        delete _tokenApprovals[tokenId];
    }
}
```

- **Safety advice**

Suggest deleting unused owner variables in the _burn method.

- **Repair Status**

The risk has been officially modified through communication with the Metaverse Player One officials.

4.2.5 No events added for sensitive operations

- **Risk description**

In order to keep users and administrators abreast of the project trends and to understand the actual operation of the contract, it is recommended that event logs be added to all methods involved by administrators and users. that shown in the following code:

```
function _transfer(address sender, address recipient, uint256 amount) internal virtual {
    require(sender != address(0), "ERC20: transfer from the zero address");
    require(recipient != address(0), "ERC20: transfer to the zero address");
    require(_balances[sender] >= amount, "ERC20: sender balance not enough");
    _balances[sender] = _balances[sender] - amount;
    _balances[recipient] = _balances[recipient] + amount;
    emit Transfer(sender, recipient, amount);
}
```

- **Safety advice**

It is recommended to add event logs for all methods involving administrators and users.

- **Repair Status**

The risk has been officially modified through communication with the Metaverse Player One officials.

4.2.6 Administrator privileges

- **Risk description**

Slot contracts, Manager contracts and other contracts where administrators can set address permissions and other sensitive operations, if the administrator's private key is controlled by a malicious person, it may lead to the loss of unusual funds and destabilise the market.. that shown in the following code:

```
function setManager(address addr) external ContractOwnerOnly {
    manager = Manager(addr);
}
function setMember(string memory name, address member) external ContractOwnerOnly {

    members[name] = member;
}

function setUserPermit(address user, string memory permit, bool enable) external
ContractOwnerOnly {

    userPermits[user][permit] = enable;
}
function setRarityExp(uint256 rarity, uint256 exp) external CheckPermit("Config") {

    for (uint256 i = rarityExps.length; i <= rarity; ++i) {
        rarityExps.push(0);
    }
    rarityExps[rarity] = exp;
}
```

- **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**

The risk has been officially modified through communication with the Metaverse Player One officials.

4.2.7 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 Findings : pass**

4.2.8 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 : pass**

4.2.9 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 : pass**

4.2.10 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 : pass**

4.2.11 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 : Passing**

4.2.12 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 : pass**

4.2.13 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 : pass**

4.2.14 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 : pass**

4.2.15 Delegatecall function call

- **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 : pass**

4.2.16 Call function 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 : pass**

4.2.17 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 : pass**

4.2.18 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 : pass**

4.2.19 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 : pass**

4.2.20 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 : pass**

4.2.21 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 Findings : pass**

4.2.22 Frozen Account passpass

- **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 : pass**

4.2.23 Contract caller not initialized

- **Risk description**

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

- **Audit results : pass**

4.2.24 Re-entry 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 : pass**

4.2.25 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 : pass**

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.io

https://github.com/lunarayio

https://twitter.com/lunarayio

http://t.me/lunarayio