LUNARAY
BLOCKCHAINSECURITY

SMART CONTRACT SECURITY AUDIT REPORT

For MetaPoint

31 October 2022

lunaray.co

Table of Contents

1. Overview

On Oct 27, 2022, the security team of Lunaray Technology received the security audit request of the **MetaPoint project**. The team completed the audit of the **MetaPoint smart contract** on Oct 31, 2022. During the audit process, the security audit experts of Lunaray Technology and the MetaPoint 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 MetaPoint 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 MetaPoint smart contract security audit: **Passed**

Audit Report Hash:
42654D2BE8B6E977A98A1BE287345E834FDF09CBA36856EFB5D7C712ABE44042

2. Background

2.1 Project Description

Project name	MetaPoint
Contract type	Token, NFT, Metaverse
Code language	Solidity
Contract file	MetaCapacity.sol, MetaFarm.sol, MetaLand.sol, MetaPoint.sol, MetaTax.sol, MinistryLandResources.sol
Introduction	MetaPoint is such a virtual online world, a prototype of the future meta-universe. In this virtual sandbox world, land is the core production resource, where land players can participate in virtual reality activities such as planting, business, construction, finance, gaming, DAO governance, and dating.

2.2 Audit Range

Smart contract file name and corresponding HASH：

Name	HASH
MetaCapacity.sol	D66E7B14BDFA129EBD1659303029E92340302B9CC9EFE3198D432BC5EF042C82
MetaFarm.sol	05E52B707010C0802CBDF6FB7339A39A790610A0A59245659FE21F75C912A44D
MetaLand.sol	4E2EB60E0EE2C221FC57B89B36B722DCA1335ACC5285D8065726DCCB3DD85DB4
MetaPoint.sol	752BBDE3EA0B005FAE66D9FFFE1C1F0535F972303888FFA1B7C36B4DD99CD89F
MetaTax.sol	91026C0DEA1D3993BD59C9151E52ADF8E4FCEE042B1FBE7023D6A55D3BBBAE8F
MinistryLandResources.sol	E7E3D885618E486344351FEE3E8967E5AA6E75498B8270C114A4288D829DB417

3. Project contract details

3.1 Contract Overview

MetaCapacity Contract

The contract acts as a capacity in the economic model and has the main function of managing the capacity data corresponding to the land.

MetaFarm Contract

The contract in the economic model mainly undertakes the estate planting lucky tree function, realize the land owner's use of the land, such as planting lucky trees (initialization of the destruction of activateToken tokens), and fertilization (use of capacity); realize the collection of their own lucky fruit (POT) and help friends to collect the function. Implemented the mechanism of deducting the helper fee for collecting from friends; implemented the functions related to managing the lucky tree of the manor, such as: setting the daily base output, maximum output, output ratio and the bonus ratio obtained by helping friends to collect, etc.

MetaLand Contract

The NFT contract, implemented according to ERC721 standard, acts as a meta-land in the economic model; it implements the functions related to casting and destruction.

MetaPoint Contract

Token contracts are implemented according to the ERC20 standard and act as lucky fruits (token symbol POT) in the economic model; mainly provide minting and destruction of POT; add Token liquidity based on PancakeSwap; and implement a blacklist mechanism.

MetaTax Contract

The contract acts as a tax point in the economic model, and the MinistryLandResources contract operates the relevant functions in the audited contract.

MinistryLandResources Contract

The contract mainly undertakes the casting of land (M-LAND NFT) in the economic model. The contract provides two methods of acquiring land, one through specific ERC20 Token exchange and the other through specific ERC20 Token purchase; implements functions related to the management of land, such as: recharging capacity of land, recharging tax points; implements a mechanism to reward the invitee when the user consumes.

3.2 Contract details

MetaLand Contract

Name	Parameter	Attributes
safeMint	address to	onlyRole
setBaseUrl	string _baseUrl	onlyRole
_beforeTokenTransfer	address from address to uint256 tokenId	internal
_burn	uint256 tokenId	internal
tokenURI	uint256 tokenId	public
supportsInterface	bytes4 interfaceId	public
_baseURI	none	internal

MetaTax Contract

Name	Parameter	Attributes
initialize	none	external
transfer	address to uint256 amount	external
transferFrom	address from address to uint256 amount	onlyRole
_transfer	address from address to uint256 amount	internal
mint	address account uint256 amount	onlyRole

MinistryLandResources Contract

Name	Parameter	Attributes
initialize	address _metaLand address _metaDeed IRandom _random IFamily _family IAchievement _achievement IPancakeRouter _swapRouter IMetaFarm _metaFarm address _metaPoint address _usdt address _metaCapacity address _metaTax	external
setReceiver	address _tax address _dao address _lab address _lpf	onlyRole
addLandPrice	uint256 _price	onlyRole
removeLandPrice	uint256 _price	onlyRole
landPriceList	none	external
addCapPrice	uint256 _price	onlyRole
removeCapPrice	uint256 _price	onlyRole
capPriceList	none	external
payTokenList	none	external
taxPayTokenList	none	external
closeMintLand	none	onlyRole
openMintLand	none	onlyRole
deedDelivery	uint256 tokenId	external
tokenAmount	address _token uint256 amount	public
mintLand	uint256 _landPrice address _token	external
recharge	address token	external

	address landAdress uint256 length uint256 _capPackPrice	
buyTax	address token uint256 value	external
pay	address _token uint256 needUSD	internal
_mintLand	address account uint256 _landPrice	internal
_mintCapacity	address tokenAddress uint256 amount bool isLand	internal
_mintTax	address account	internal
distributeReward	address account uint256 amount	internal
_shareManure	address provider address recipient uint256 value	internal

MetaPoint Contract

Name	Parameter	Attributes
setSellFee	uint256 _sellFee	onlyOwner
setBuyFee	uint256 _buyFee	onlyOwner
initPOTLiquidity	address tokenA address tokenB uint256 amountADesired uint256 amountBDesired uint256 amountAMin uint256 amountBMin address to uint256 deadline	external
_transfer	address from address to uint256 amount	internal
_addBlackUser	address account	internal

MetaFarm Contract

Name	Parameter	Attributes
initialize	IERC20Upgradeable _token IERC20Upgradeable _depositToken IERC20Upgradeable _activateToken IERC721Enumerable _metaLand IFriend _friend IPancakeRouter _swapRouter IAchievement _achievement IERC20Upgradeable _taxToken address _usdt	public
addUserPayUSD	address account uint256 amount	onlyRole
addDayPotTotalBurn	uint256 dayTime uint256 amount	onlyRole
addUserMaxEarnUSD	address account uint256 amount	onlyRole
updatePool	none	public
getDayOutPut	uint256 dayStartTime	public
setMaxDayOutPut	uint256 _maxDayOutPut	onlyRole
setBasicsDayOutPut	uint256 _basicsDayOutPut	onlyRole
setOutPutRadio	uint256 _radio	onlyRole
setFriendHarvestRadio	uint256 _friendHarvestRadio	onlyRole
setActiveTokenNeed	uint256 _activeTokenNeed	onlyRole
setTaxRadio	uint256 _lowTaxRadio uint256 _bigTaxRadio	onlyRole
start	none	onlyRole
getTokenPerShare	uint256 _time	internal
earned	address _landAddress	internal
cultivate	address _landAddress	external
fertilization	uint256 _amount address _landAddress address from	onlyRole

_deposit	uint256 _amount address _landAddress bool isUpdateTime	internal
earnValue	uint256 amount	public
harvest	address _landAddress	external
harvestInfo	address _landAddress	external

MetaCapacity Contract

Name	Parameter	Attributes
initialize	IERC721Enumerable _metaLand	external
setFarm	IMetaFarm _metaFarm	onlyRole
transfer	address to uint256 amount	external
transferFrom	address from address to uint256 amount	onlyRole
_transfer	address from address to uint256 amount	internal
mint	address account uint256 amount	onlyRole

4. Audit details

4.1 Findings Summary

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

4.2 Risk distribution

Name	Risk level	Repair status
Administrator Permissions	Info	Acknowledged
Redundant codes	Info	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

Integer Overflow	No	normal

4.3 Risk audit details

4.3.1 Administrator Permissions

- **Risk description**

Currently in the contract, only the Owner administrator can set contract-related parameters, which may affect the stability of the project market when the administrator is maliciously manipulated or the private key is leaked.

```
function setMaxDayOutPut(uint256 _maxDayOutPut)
    external
    onlyRole(MANAGER_ROLE)
{
    require(_maxDayOutPut > 0, "MetaFarm: maxDayOutPut must ge 0");
    updatePool();
    maxDayOutPut = _maxDayOutPut;
}
function setBasicsDayOutPut(uint256 _basicsDayOutPut)
    external
    onlyRole(MANAGER_ROLE)
{
    require(_basicsDayOutPut > 0, "MetaFarm: basicsDayOutPut must ge 0
");
    updatePool();
    basicsDayOutPut = _basicsDayOutPut;
}
function setOutPutRadio(uint256 _radio) external onlyRole(MANAGER_ROLE)
 {
    updatePool();
    outPutRadio = _radio;
}
function setFriendHarvestRadio(uint256 _friendHarvestRadio)
    external
    onlyRole(MANAGER_ROLE)
{
    require(
        _friendHarvestRadio < 1e12,
        "MetaFarm: friendHarvestRadio must lt 1e12"
    );
    friendHarvestRadio = _friendHarvestRadio;
}
function setActiveTokenNeed(uint256 _activeTokenNeed)
    external
    onlyRole(MANAGER_ROLE)
{
```

```
    activeTokenNeed = _activeTokenNeed;
}
function setTaxRadio(uint256 _lowTaxRadio, uint256 _bigTaxRadio)
    external
    onlyRole(MANAGER_ROLE)
{
    require(
        _lowTaxRadio < _bigTaxRadio,
        "MetaFarm: lowTaxRadio must lt bigTaxRadio"
    );
    require(_bigTaxRadio < 1e12, "MetaFarm: bigTaxRadio must lt 1e12");

    lowTaxRadio = _lowTaxRadio;
    bigTaxRadio = _bigTaxRadio;
}
```

- **Safety advice**

It is recommended to use multi-signature contracts to control administrator privileges.

- **Repair Status**

MetaPoint has confirmed.

4.3.2 Redundant codes

- **Risk description**

The presence of meaningless or never-executing code in the contract code may cause unnecessary consumption of fees.

```
function harvestInfo(address _landAddress)
        external
        returns (uint256,uint256,uint256)
{
    require(msg.sender == address(0xdead),'only call');
    updatePool();
    address owner = metaLand.ownerOf(uint256(uint160(_landAddress)));
    uint256 reward = earned(_landAddress);
    uint256 rewardValue = reward > 0 ? earnValue(reward) : 0;
    uint256 tmpTaxRadio;
    if (userTakeUSD[owner] > userPayUSD[owner] * 2) {
        tmpTaxRadio = bigTaxRadio;
    } else {
        tmpTaxRadio = lowTaxRadio;
    }
    return (reward, rewardValue, (rewardValue * tmpTaxRadio) / 1e12);
}
```

- **Safety advice**

Remove code that can never be called or makes no sense to avoid incurring addition al fees.

- **Repair Status**

MetaPoint has confirmed.

4.3.3 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**

4.3.4 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**

4.3.5 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**

4.3.6 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**

4.3.7 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**

4.3.8 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**

4.3.9 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**

4.3.10 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**

4.3.11 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**

4.3.12 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**

4.3.13 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**

4.3.14 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**

4.3.15 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**

4.3.16 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**

4.3.17 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**

4.3.18 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**

4.3.19 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**

4.3.20 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**

4.3.21 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**

4.3.22 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.

https://lunaray.co

https://github.com/lunaraySec

https://twitter.com/lunaray_Sec

http://t.me/lunaraySec