SMART CONTRACT SECURITY AUDIT REPORT

for xParallel Space

21 December 2021

lunaray.co

Table of Contents

1. Overview

On Nov 14, 2021, the security team of Lunaray Technology received the security audit request of the **xParallel space project**. The team completed the audit of the **xParallel space smart contract** on Dec 21, 2021. During the audit process, the security audit experts of Lunaray Technology and the xParallel space 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 xParallel space 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 xParallel space smart contract security audit:

Passed

Audit Report Hash:

29CD407FAC17D02874BC3815CE7DF6C941B280FFF6CD4B74FDE941C33376FC70

2. Background

2.1 Project Description

Project name	xParallel space
Contract type	Token , DeFi
Total Issue	2 000 000 (XPS)
Code language	Solidity
Public chain	Binance Smart Chain
Project address	https://xpslab.com/
Contract file	XPS.sol, LockPlan.sol, Burn.sol, StakingRewards_XPS.sol, Miner.sol
Project Description	xParallel space is a chain-wide interoperable overborrowing and lending smart protocol that allows users to participate in liquidity mining by overborrowing users can participate in liquidity mining by overborrowing and leveraging, i.e., they can earn higher returns per unit of time.

2.2 Audit Range

The smart contract file provided by xParallel space and the corresponding MD5：

Name	address
Burn.sol	BF1FE704ABA3E55B063F190D838F867C
LockPlan.sol	8B023FDBD7179C0BE3DFFB722BAA6BEE
Miner.sol	B61A328FCA9770D62094A586DE15F60D
StakingRewards_XPS.sol	AC96B189854510CD30AE52400ADAEF7A
XPS.sol	639313A8531AE23EAEF5D2FC5C32F3B1

3. Project contract details

3.1 Contract Overview

XPS Contract

The XPS Token section manages the casting and destruction of all XPS Tokens.

LockPlan Contract

Manage user lockout positions, create lockout plans and close lockout plans.

Burn Contract

Set up a pledge contract with the main function of destroying the XPS Token in the caller's address.

Miner Contract

Mining contract, set miner address, start time and last update time, can be block out through mining interface.

StakingRewards_XPS Contract

Project master contract, increase arithmetic power by pledging funds, can get XPS Token reward, can remove funds to exit arithmetic power, miner interface is used here.

3.2 Contract details

Burn Contract

Name	Parameter	Attributes
Burn	IPair _xpsLP	default
setStaking	IStakingRewards _stakingRewards	onlyOwner
circleOfBurn	uint256 _amount	public
burn	uint256 _xpsAmount	external

LockControl Contract

Name	Parameter	Attributes
LockControl	IERC20 _lockToken	default
_addPlans	address _plan	private
creatPlan	none	onlyOwner
closeAddress	uint256 _pId address _user uint256 _id	onlyOwner

XPS Contract

Name	Parameter	Attributes
mint	address _to uint256 _amount	onlyMinter
burn	uint256 _amount	external

LockPlan Contract

Name	Parameter	Attributes
LockPlan	address _lockToken	default
planLength	address _owner	external
getPlansOfOwner	address _owner	external
_claim	address _owner	private
claim	address _owner	external
closePlan	address _user uint256 _id	onlyOwner

Miner Contract

Name	Parameter	Attributes
Miner	address _earnToken uint256 _binary	default
end	none	public
setEarn	address _miner uint256 _start uint256 _lastMiningTime	onlyOwner
getIncrementEarn	uint256 _endTime	public
getTotalIncrementEarn	uint256 _end	public
mining	none	external
min	uint256 a uint256 b	public
kill	none	onlyOwner

XPSFactory Contract

Name	Parameter	Attributes
setMiner	IMeerFactory _miner	onlyOwner
setLock	bool _lock	onlyOwner
setBurn	IBurnProxy _burnProxy	onlyOwner
setCircle	address _owner uint256 _circleEPX	onlyBurnProxy
balanceOf	address _owner	external
balanceShareOf	address _owner	external
balanceReferrer20Of	address _owner	external
energyOfOwner	address _owner uint256 _indexs	external
initToArray	uint256 _indexs uint256 _len	public
getActiveOfOwner	address _owner	external
_setCircle	address _owner uint256 _circleEPX	internal
_initCircle	address _owner	internal
_upDateOwnerEarn	address _owenr	internal
_upDateBeforeEarn TokenBalance	none	internal
calNet	none	public
claimXPSFor	address _owner	external
_setOldEnergyAmountEPX	Energy bal	internal
_initOldEnergyAmountEPX	none	internal
_setNewEnergyAmountEPX	Energy bal	internal
_initNweEnergyAmountEPX	none	internal
_close	Energy bal	internal

_isAdd	none	internal
_unstakeXPSFor	address _owner	internal
_stakeXpsFor	address _owner uint256 _xps	internal
_setBoost	Energy _bal uint256 _type	internal
lockBoostTime	uint256 _type	public
circleEPXFor	address _owner	public
circleOfBurnEPX	uint256 _circleEPX	public
_beforeCalNet	none	internal

XPSWarehouse Contract

Name	Parameter	Attributes
XPSWarehouse	address _admin	default
withdraw	address token uint256 amount	onlyOwner
setAdmin	address _admin	onlyAdmin
receiveERC20	address token	onlyAdmin

4. Audit details

4.1 Findings Summary

Severity	Found	Resolved	Acknowledged
High	0	0	0
Medium	0	0	0
Low	3	1	2
Info	9	1	8

4.2 Risk distribution

Name	Risk level	Repair status
Administrator Permissions	Low	Acknowledged
Address validity not determined	Info	Acknowledged
No events added	Info	Acknowledged
The status change occurs after the transfer	Low	Resolved
Incorrect storage pointer initialization	Info	Acknowledged
Interface logic and security unknown	Info	Acknowledged
The calNet method is called multiple times	Info	Acknowledged
_upDateOwnerEarn method call	Info	Acknowledged
Redundant codes	Info	Resolved
Increasing the risk of rewards	Low	Acknowledged
Duplicate values	Info	Acknowledged
Code definition before the contract	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**

StakingRewards contract, withdraw method, receiveERC20 method can perform sensitive operations, if the administrator's private key is controlled by malicious people, it may lead to abnormal money loss and shake the stability of the market, as shown in the following code:

```
    function withdraw(address token, uint256 amount) external onlyOwner
 {
        token.safeTransfer(msg.sender, amount);
    }

    function receiveERC20(address token) external onlyAdmin {
        uint256 balance = token.myBalance();
        token.safeTransfer(msg.sender, balance);
    }
```

- **Safety advice**

recommend setting TimeLock time locks for time constraints on administrator operations, Properly store the administrator's private key to ensure the storage security of the private key.

- **Repair Status**

This risk has been identified by xParallel space.

4.3.2 Address validity not determined

- **Risk description**

StakingRewards contract, setAdmin method, can be modified onlyAdmin modifier in the administrator admin address, if the administrator private key is controlled by malicious people or unintentional operations lead to admin updated to 0 address, after changing the modifier to participate in the method can no longer be called, eventually leading to abnormal capital loss and shake the stability of the market, as follows Code shown:

```
function setAdmin(address _admin) external onlyAdmin {
    admin = _admin;
}
```

- **Safety advice**

It is recommended that the setAdmin method determine that the _admin address cannot be a zero address.

- **Repair Status**

This risk has been identified by xParallel space.

4.3.3 No events added

- **Risk description**

StakingRewards contract, withdraw method, receiveERC20 method, addEnergy method, removeEnergy method, claimXPSFor method and many other methods will perform sensitive operations, in order to keep users and administrators informed of the contract operation details, it is recommended to add event logging, the following code shown below:

```
function removeEnergy(uint256 _indexs, uint256[20] calldata _actives) e
xternal {
 require(unlock || _balanceEnergy[msg.sender].lockEnd <= block.timestam
p, "Not yet expired");
 require(_balanceEnergy[msg.sender].balance>0,"Insufficient balance");
        calNet();
        _upDateOwnerEarn(msg.sender);
        Energy storage bal = _balanceEnergy[msg.sender];
        uint256 unstakeAmount = bal.balance;
        _setOldEnergyAmountEPX(bal);
        _unstakeXPSFor(msg.sender);
        _setNewEnergyAmountEPX(bal);
        _changeEnergyFor(msg.sender, unstakeAmount,_indexs, _actives);
        _close(bal);
        _initNweEnergyAmountEPX();
        _initNweEnergyAmountEPX();
    }
function claimXPSFor(address _owner) external returns(uint256 rewrods)
{       calNet();
        rewrods = _upDateOwnerEarn(_owner);
        uint256 balance = xps.myBalance();
        if ( balance < rewrods ) {
            emit Claim(balance, rewrods);
        }
        rewrods = Math.min(balance, rewrods);
        xps.safeTransfer(_owner, rewrods);
        _balanceEnergy[_owner].debtEarn -= rewrods;
        _upDateBeforeEarnTokenBalance();
    }
```

- **Safety advice**

Suggest adding event logging for sensitive operations.

- **Repair Status**

This risk has been identified by xParallel space.

4.3.4 The status change occurs after the transfer

- **Risk description**

StakingRewards contract, claimXPSFor method, balanceEnergy[owner].debtEarn variable value change occurs after the transfer, as shown in the following code:

```
    function claimXPSFor(address _owner) external returns(uint256 rewro
ds) {
        calNet();
        rewrods = _upDateOwnerEarn(_owner);

        uint256 balance = xps.myBalance();

        if ( balance < rewrods ) {
            emit Claim(balance, rewrods);
        }

        rewrods = Math.min(balance, rewrods);
        xps.safeTransfer(_owner, rewrods);
        _balanceEnergy[_owner].debtEarn -= rewrods;
        _upDateBeforeEarnTokenBalance();
    }
```

- **Safety advice**

To avoid reentrancy, the variable balanceEnergy[owner].debtEarn is recommended to be written before the transfer.

- **Repair Status**

The risk has been fixed in xParallel space.

4.3.5 Incorrect storage pointer initialization

- **Risk description**

XPSFactory contract, addEnergy, removeEnergy methods, Energy type structure bal without state change. The way Solidity works, state variables are stored in the Slot of the contract in the order they appear in the contract, and uninitialized local storage variables may point to other accidental storage variables in the contract, leading to intentional or unintentional vulnerabilities. Also, using storage increases the use of gas for executing the contract, increasing the risk of the contract being rolled back due to excessive gas consumption, as shown in the following code:

```
   function addEnergy(uint256 _xpsAmount, uint256 _lockType, uint256 _i
ndexs, uint256[20] calldata _actives) external {
        (address _owner,) = relation.referrer(msg.sender);
        require(_owner != address(0), "sender no referrer");
        _initCircle(msg.sender);
        calNet();
        _upDateOwnerEarn(msg.sender);

        Energy storage bal = _balanceEnergy[msg.sender];use of storage
        _setOldEnergyAmountEPX(bal);
        _stakeXpsFor(msg.sender , _xpsAmount);
        _setBoost(bal, _lockType);
        _setNewEnergyAmountEPX(bal);

        _changeEnergyFor(msg.sender, _xpsAmount, _indexs, _actives);

        _initNweEnergyAmountEPX();
        _initNweEnergyAmountEPX();
    }

    function removeEnergy(uint256 _indexs, uint256[20] calldata _active
s) external {
        require(unlock || _balanceEnergy[msg.sender].lockEnd <= block.t
imestamp, "Not yet expired");
        require(_balanceEnergy[msg.sender].balance > 0,"Insufficient ba
lance");
        calNet();
        _upDateOwnerEarn(msg.sender);

        Energy storage bal = _balanceEnergy[msg.sender];
        uint256 unstakeAmount = bal.balance;
        _setOldEnergyAmountEPX(bal);
        _unstakeXPSFor(msg.sender);
        _setNewEnergyAmountEPX(bal);
```

```
        _changeEnergyFor(msg.sender, unstakeAmount,_indexs, _actives);

        _close(bal);
        _initNweEnergyAmountEPX();
        _initNweEnergyAmountEPX();
    }
```

- **Safety advice**

It is recommended to use memory instead of storage to initialize the structure.

- **Repair Status**

This risk has been identified by xParallel space.

4.3.6 Interface logic and security unknown

- **Risk description**

StakingRewards contract, _initCircle method, addEnergy method, balanceReferrer20Of method and many other methods are called in the external interface methods, currently can not understand the interface logic and security, the following code is shown:

```
    function _initCircle(address _owner) internal {
        Energy storage enr = _balanceEnergy[_owner];
        if ( enr.initCircle == false ) {
            _setCircle(_owner,burnProxy.circleEPXFor(_owner));
            enr.initCircle = true;
        }
    }

    function addEnergy(uint256 _xpsAmount, uint256 _lockType, uint256 _
indexs, uint256[20] calldata _actives) external {
        (address _owner,) = relation.referrer(msg.sender);
        require(_owner != address(0), "sender no referrer");
        _initCircle(msg.sender);
        calNet();
        _upDateOwnerEarn(msg.sender);

        Energy storage bal = _balanceEnergy[msg.sender];
        _setOldEnergyAmountEPX(bal);
        _stakeXpsFor(msg.sender, _xpsAmount);
        _setBoost(bal, _lockType);
        _setNewEnergyAmountEPX(bal);

        _changeEnergyFor(msg.sender, _xpsAmount, _indexs, _actives);

        _initNweEnergyAmountEPX();
        _initNweEnergyAmountEPX();
    }
```

- **Safety advice**

At present, it is impossible to understand the logic and security of the interface called, and it is recommended that the official self-investigate the logic and security of the external interface.

- **Repair Status**

This risk has been identified by xParallel space.

4.3.7 The calNet method is called multiple times

- **Risk description**

S StakingRewards contract, addEnergy method third line call initCircle method, when the user first call initCircle method, the method will subsequently call the calNet method, in the addEnergy method fourth line, also called the calNet method, the method called twice, through analysis found that the user pledge funds seems to be unnecessary to call the calNet method twice to get the total amount of rewards token in the current contract, as shown in the following code:

```
    function addEnergy(uint256 _xpsAmount, uint256 _lockType, uint256 _
indexs, uint256[20] calldata _actives) external {
        (address _owner,) = relation.referrer(msg.sender);
        require(_owner != address(0), "sender no referrer");
        _initCircle(msg.sender);
        calNet();
        _upDateOwnerEarn(msg.sender);

        Energy storage bal = _balanceEnergy[msg.sender];
        _setOldEnergyAmountEPX(bal);
        _stakeXpsFor(msg.sender, _xpsAmount);
        _setBoost(bal, _lockType);
        _setNewEnergyAmountEPX(bal);

        _changeEnergyFor(msg.sender, _xpsAmount, _indexs, _actives);

        _initNweEnergyAmountEPX();
        _initNweEnergyAmountEPX();
    }

    function _initCircle(address _owner) internal {
        Energy storage enr = _balanceEnergy[_owner];
        if ( enr.initCircle == false ) {
            _setCircle(_owner,burnProxy.circleEPXFor(_owner));
            enr.initCircle = true;
        }
    }

    function _setCircle(address _owner, uint256 _circleEPX) internal {
        Energy storage enr = _balanceEnergy[_owner];
        require(_circleEPX >= enr.circleEPX, "_circleEPX too low");
        calNet();
        _upDateOwnerEarn(_owner);
        uint256 addEnr = (_circleEPX - enr.circleEPX) * enr.totalShare
/ EPX;
        enr.circleEPX = _circleEPX;
```

```
        totalEnergy += addEnr;
        _balanceEnergy[_owner].circleEPX = _circleEPX;
        _balanceEnergy[_owner].initCircle = true;
    }
```

- **Safety advice**

If there is no need to call the calNet method twice, it is recommended to delete one of them to save Gas.

- **Repair Status**

This risk has been identified by xParallel space.

4.3.8 _upDateOwnerEarn method call

- **Risk description**

StakingRewards contract, the third line of the addEnergy method calls the initCircle method,, and when the user calls the initCircle method for the first time, the method subsequently calls the upDateOwnerEarn method, and in the fifth line of the addEnergy method, it also calls the upDateOwnerEarn method, which is called twice, as shown in the following code:

```
    function addEnergy(uint256 _xpsAmount, uint256 _lockType, uint256 _
indexs, uint256[20] calldata _actives) external {
        (address _owner,) = relation.referrer(msg.sender);
        require(_owner != address(0), "sender no referrer");
        _initCircle(msg.sender);
        calNet();
        _upDateOwnerEarn(msg.sender);

        Energy storage bal = _balanceEnergy[msg.sender];
        _setOldEnergyAmountEPX(bal);
        _stakeXpsFor(msg.sender, _xpsAmount);
        _setBoost(bal, _lockType);
        _setNewEnergyAmountEPX(bal);

        _changeEnergyFor(msg.sender, _xpsAmount, _indexs, _actives);

        _initNweEnergyAmountEPX();
        _initNweEnergyAmountEPX();
    }

    function _initCircle(address _owner) internal {
        Energy storage enr = _balanceEnergy[_owner];
        if ( enr.initCircle == false ) {
            _setCircle(_owner,burnProxy.circleEPXFor(_owner));
            enr.initCircle = true;
        }
    }

    function _setCircle(address _owner, uint256 _circleEPX) internal {
        Energy storage enr = _balanceEnergy[_owner];
        require(_circleEPX >= enr.circleEPX, "_circleEPX too low");
        calNet();
        _upDateOwnerEarn(_owner);
        uint256 addEnr = (_circleEPX - enr.circleEPX) * enr.totalShare
/ EPX;
        enr.circleEPX = _circleEPX;
        totalEnergy += addEnr;
```

```
        _balanceEnergy[_owner].circleEPX = _circleEPX;
        _balanceEnergy[_owner].initCircle = true;
    }
```

- **Safety advice**

If there is no need to call _upDateOwnerEarn(msg.sender) twice, it is recommended to delete one of them to save Gas.

- **Repair Status**

This risk has been identified by xParallel space.

4.3.9 Redundant codes

- **Risk description**

StakingRewards contract, addEnergy method, removeEnergy method, _initNweEnergyAmountEPX() the method was called twice respectively, as shown in the following code:

```
    function addEnergy(uint256 _xpsAmount, uint256 _lockType, uint256 _
indexs, uint256[20] calldata _actives) external {
        (address _owner,) = relation.referrer(msg.sender);
        require(_owner != address(0), "sender no referrer");
        _initCircle(msg.sender);
        calNet();
        _upDateOwnerEarn(msg.sender);
        Energy storage bal = _balanceEnergy[msg.sender];
        _setOldEnergyAmountEPX(bal);
        _stakeXpsFor(msg.sender, _xpsAmount);
        _setBoost(bal, _lockType);
        _setNewEnergyAmountEPX(bal);

        _changeEnergyFor(msg.sender, _xpsAmount, _indexs, _actives);

        _initNweEnergyAmountEPX();

        _initNweEnergyAmountEPX();
    }

    function removeEnergy(uint256 _indexs, uint256[20] calldata _active
s) external {
        ……
        _initNweEnergyAmountEPX();
        _initNweEnergyAmountEPX();
    }
```

- **Safety advice**

If there is no need to call _initNweEnergyAmountEPX() twice, it is recommended to delete one of them to save Gas.

- **Repair Status**

The risk has been fixed in xParallel space.

4.3.10 Increasing the risk of rewards

- **Risk description**

The StakingRewards contract, through the claimXPSFor method to receive the proceeds, claimXPSFor method will call the following three methods in turn:

calNet()-----_upDateOwnerEarn()-----safeTransfer()

The above three methods content algorithms are mainly as follows:

1. calNet()：Calculate incremental revenue

2. _upDateOwnerEarn()：Calculate the final gain obtained by the user (the gain increment cumulativeNetWorth global variable is used in the calculation)

The final revenue algorithm obtained by the user is as follows:

```
enr.debtEarn += (enr.balance * enr.boostEPX / EPX + enr.totalShare * en
r.circleEPX / EPX) * ( cumulativeNetWorth - enr.net ) / EPX2;
```

It is clear from the above algorithm that (cumulativeNetWorth - enr.net) / EPX2 algorithm, when EPX2 and enr.net variable value remains unchanged, the increase of cumulativeNetWorth will make the final result of the whole algorithm increase, and it is known that EPX2 value will not change, enr.net value will be updated when the current _ The cumulativeNetWorth variable can be obtained by the following algorithm in the calNet() method: cumulativeNetWorth += EPX2 * earnAmount / totalEnergy; and after each call to the calNet() method The value of the cumulativeNetWorth variable is incremented after each call to the calNet() method.

3. safeTransfer()：Sending revenue to users

Through the above analysis, we can conclude that when the user pledges funds and gets the gain, he can first call the calNet() method several times to make the global variable of cumulativeNetWorth increasing and the denominator in the gain algorithm increasing, and afterwards call the claimXPSFor method to receive a large amount of rewards.

The method call is shown in the following code：

```
First call the pledged funds addEnergy method, so that the user custome
r gets the revenue
    function addEnergy(uint256 _xpsAmount, uint256 _lockType, uint256 _
indexs, uint256[20] calldata _actives) external {
        (address _owner,) = relation.referrer(msg.sender);
        require(_owner != address(0), "sender no referrer");
        _initCircle(msg.sender);
        calNet();
```

```
        _upDateOwnerEarn(msg.sender);
        Energy storage bal = _balanceEnergy[msg.sender];
        _setOldEnergyAmountEPX(bal);
        _stakeXpsFor(msg.sender, _xpsAmount);
        _setBoost(bal, _lockType);
        _setNewEnergyAmountEPX(bal);
        _changeEnergyFor(msg.sender, _xpsAmount, _indexs, _actives);
        _initNweEnergyAmountEPX();
        _initNweEnergyAmountEPX();
    }
```

```
Secondly, the calNet method is called several times to increase the val
ue of the global variable cumulativeNetWorth
    function calNet() public {
        _beforeCalNet();
        if ( totalEnergy > 0 ) {
            uint256 newEarn = xps.myBalance();
            uint256 earnAmount = newEarn - beforeEarnTokenBalance;
            cumulativeNetWorth += EPX2 * earnAmount / totalEnergy;
            _upDateBeforeEarnTokenBalance();
        }
    }
After that, the claimXPSFor method will be called to collect the revenu
e, and the increased global variable cumulativeNetWorth will be brought
 into the calculation to get a larger reward value to achieve more reve
nue.
    function claimXPSFor(address _owner) external returns(uint256 rewro
ds) {
        calNet();
        rewrods = _upDateOwnerEarn(_owner);
        uint256 balance = xps.myBalance();
        if ( balance < rewrods ) {
            emit Claim(balance, rewrods);
        }
        rewrods = Math.min(balance, rewrods);
        xps.safeTransfer(_owner, rewrods);
        _balanceEnergy[_owner].debtEarn -= rewrods;
        _upDateBeforeEarnTokenBalance();
    }
```

- **Safety advice**

If there is an impact, it is recommended that the variables used to calculate the reward be strictly limited.

- **Repair Status**

This risk has been identified by xParallel space.

4.3.11 Duplicate values

- **Risk description**

StakingRewards contract, _setCircle method, the code takes repeated values, as shown in the following code:

```
function _setCircle(address _owner, uint256 _circleEPX) internal {
   Energy storage enr = _balanceEnergy[_owner];

    require(_circleEPX >= enr.circleEPX, "_circleEPX too low");
    calNet();
    _upDateOwnerEarn(_owner);
    uint256 addEnr = (_circleEPX - enr.circleEPX) * enr.totalShare / EP
X;
    enr.circleEPX = _circleEPX;
    totalEnergy += addEnr;
    _balanceEnergy[_owner].circleEPX = _circleEPX;
    _balanceEnergy[_owner].initCircle = true;
}
```

- **Safety advice**

It is recommended to use enr to modify directly, without taking the value again.

- **Repair Status**

This risk has been identified by xParallel space.

4.3.12 Code definition before the contract

- **Risk description**

StakingRewards contract, structure EnergyValue, some of the value definitions are displayed before the methods that use its code, it is recommended that the above definitions are written before the contract to avoid errors when the methods are called after changes, as shown in the following code:

```
struct EnergyValue {
    uint256 totalAddEnergyEPX;
    uint256 totalSubEnergyEPX;
    uint256 energyAmountShareEPX;
    uint256 energyAmountNetShareEPX;
}

uint256[] public lockTime = [30 days,90 days,180 days,360 days];
uint256[] public lockTimeBoostEpx = [100000,200000,500000,1000000];
```

- **Safety advice**

It is recommended that the above definitions are written before the contract to make it easier to call.

- **Repair Status**

This risk has been identified by xParallel space.

4.3.13 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**

4.3.14 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**

4.3.15 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**

4.3.16 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**

4.3.17 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**

4.3.18 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**

4.3.19 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**

4.3.20 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**

4.3.21 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**

4.3.22 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**

4.3.23 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**

4.3.24 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**

4.3.25 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**

4.3.26 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**

4.3.27 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**

4.3.28 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**

4.3.29 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**

4.3.30 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**

4.3.31 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**

4.3.32 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