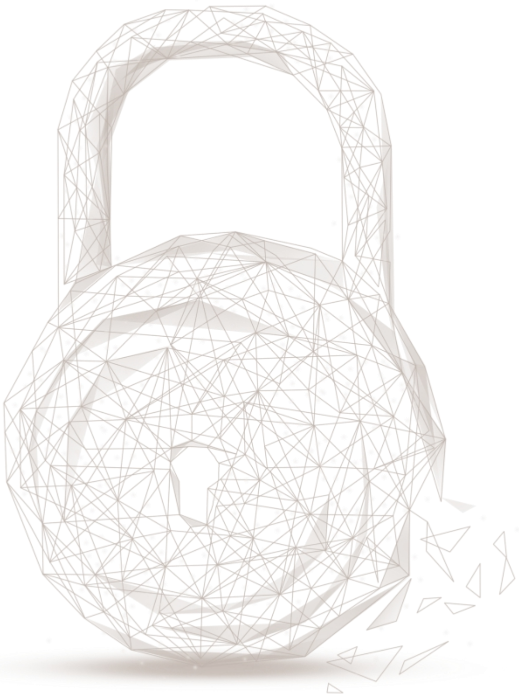
BEOSIN
Blockchain Security

Smart contract security audit report



Audit Number：202104072033

Report Query Name：AIOZ

Audit Project Name：AIOZ

Smart Contract Address：

0x626E8036dEB333b408Be468F951bdB42433cBF18

Smart Contract Address Link：

https://etherscan.io/address/0x626e8036deb333b408be468f951bdb42433cbf18#code

Start Date：2021.04.01

Completion Date：2021.04.07

Overall Result：Pass

Audit Team: Beosin (Chengdu LianAn) Technology Co. Ltd.

Audit Categories and Results:

No.	Categories	Subitems	Results
1	Coding Conventions	Compiler Version Security	Pass
		Deprecated Items	Pass
		Redundant Code	Pass
		SafeMath Features	Pass
		require/assert Usage	Pass
		Gas Consumption	Pass
		Visibility Specifiers	Pass
		Fallback Usage	Pass
2	General Vulnerability	Integer Overflow/Underflow	Pass
		Reentrancy	Pass
		Pseudo-random Number Generator (PRNG)	Pass
		Transaction-Ordering Dependence	Pass
		DoS (Denial of Service)	Pass
		Access Control of Owner	Pass

		Low-level Function (call/delegatecall) Security	Pass
		Returned Value Security	Pass
		tx.origin Usage	Pass
		Replay Attack	Pass
		Overriding Variables	Pass
3	Business Security	Business Logics	Pass
		Business Implementations	Pass

Disclaimer: This audit is only applied to the type of auditing specified in this report and the scope of given in the results table. Other unknown security vulnerabilities are beyond auditing responsibility. Beosin (Chengdu LianAn) Technology only issues this report based on the attacks or vulnerabilities that already existed or occurred before the issuance of this report. For the emergence of new attacks or vulnerabilities that exist or occur in the future, Beosin (Chengdu LianAn) Technology lacks the capability to judge its possible impact on the security status of smart contracts, thus taking no responsibility for them. The security audit analysis and other contents of this report are based solely on the documents and materials that the contract provider has provided to Beosin (Chengdu LianAn) Technology before the issuance of this report, and the contract provider warrants that there are no missing, tampered, deleted; if the documents and materials provided by the contract provider are missing, tampered, deleted, concealed or reflected in a situation that is inconsistent with the actual situation, or if the documents and materials provided are changed after the issuance of this report, Beosin (Chengdu LianAn) Technology assumes no responsibility for the resulting loss or adverse effects. The audit report issued by Beosin (Chengdu LianAn) Technology is based on the documents and materials provided by the contract provider, and relies on the technology currently possessed by Beosin (Chengdu LianAn). Due to the technical limitations of any organization, this report conducted by Beosin (Chengdu LianAn) still has the possibility that the entire risk cannot be completely detected. Beosin (Chengdu LianAn) disclaims any liability for the resulting losses.

The final interpretation of this statement belongs to Beosin (Chengdu LianAn).

Audit Results Explained:

Beosin (Chengdu LianAn) Technology has used several methods including Formal Verification, Static Analysis, Typical Case Testing and Manual Review to audit three major aspects of smart contract AIOZ, including Coding Standards, Security, and Business Logic. **The AIOZ contract passed all audit items. The overall result is Pass.** The smart contract is able to function properly.

Audit Contents:

1. Coding Conventions

Check the code style that does not conform to Solidity code style.

1.1 Compiler Version Security

- Description: Check whether the code implementation of current contract contains the exposed solidity compiler bug.

  The smart contract specifies the 0.8.3 version of the compiler to compile the contract on the main network, and the contract is compiled with this version of the compiler without any compiler warning.
- Safety Suggestion: None
- Result: Pass

1.2 Deprecated Items

- Description: Check whether the current contract has the deprecated items.
- Safety Suggestion: None
- Result: Pass

1.3 Redundant Code

- Description: Check whether the contract code has redundant codes.
- Safety Suggestion: None
- Result: Pass

1.4 SafeMath Features

- Description: Check whether the SafeMath has been used. Or prevents the integer overflow/underflow in mathematical operation.
- Safety Suggestion: None
- Result: Pass

1.5 require/assert Usage

- Description: Check the use reasonability of 'require' and 'assert' in the contract.
- Safety Suggestion: None
- Result: Pass

1.6 Gas Consumption

- Description: Check whether the gas consumption exceeds the block gas limitation.
- Safety Suggestion: None
- Result: Pass

1.7 Visibility Specifiers

- Description: Check whether the visibility conforms to design requirement.
- Safety Suggestion: None
- Result: Pass

1.8 Fallback Usage

- Description: Check whether the Fallback function has been used correctly in the current contract.
- Safety Suggestion: None

- Result: Pass

2. General Vulnerability

Check whether the general vulnerabilities exist in the contract.

2.1 Integer Overflow/Underflow

- Description: Check whether there is an integer overflow/underflow in the contract and the calculation result is abnormal.
- Safety Suggestion: None
- Result: Pass

2.2 Reentrancy

- Description: An issue when code can call back into your contract and change state, such as withdrawing ETH.
- Safety Suggestion: None
- Result: Pass

2.3 Pseudo-random Number Generator (PRNG)

- Description: Whether the results of random numbers can be predicted.
- Safety Suggestion: None
- Result: Pass

2.4 Transaction-Ordering Dependence

- Description: Whether the final state of the contract depends on the order of the transactions.
- Safety Suggestion: None
- Result: Pass

2.5 DoS (Denial of Service)

- Description: Whether exist DoS attack in the contract which is vulnerable because of unexpected reason.
- Safety Suggestion: None
- Result: Pass

2.6 Access Control of Owner

- Description: Whether the owner has excessive permissions, such as malicious issue, modifying the balance of others.
- Safety Suggestion: None
- Result: Pass

2.7 Low-level Function (call/delegatecall) Security

- Description: Check whether the usage of low-level functions like call/delegatecall have vulnerabilities.
- Safety Suggestion: None

- Result: Pass

2.8 Returned Value Security

- Description: Check whether the function checks the return value and responds to it accordingly.
- Safety Suggestion: None
- Result: Pass

2.9 tx.origin Usage

- Description: Check the use secure risk of 'tx.origin' in the contract.
- Safety Suggestion: None
- Result: Pass

2.10 Replay Attack

- Description: Check the weather the implement possibility of Replay Attack exists in the contract.
- Safety Suggestion: None
- Result: Pass

2.11 Overriding Variables

- Description: Check whether the variables have been overridden and lead to wrong code execution.
- Safety Suggestion: None
- Result: Pass

3. Business Audit

3.1 The AIOZToken Contract Audit

3.1.1 Contract owner permission management

- Description: The manager permission owner of this contract (the contract deployer by default) can call the *transferOwnership* function to transfer the owner permission to the specified non-zero address; or call the *renounceOwnership* function to renounce the owner permission; call the *mint* function to mint tokens to the specified address; call the *burn* function to burn tokens.
- Related functions: *transferOwnership, renounceOwnership, mint, burn*
- Result: Pass

3.1.2 Basic information of AIOZ token

- Description: The AIOZToken contract implements an ERC20 token, and its basic information is as follows:

Token name	AIOZ Network
Token symbol	AIOZ
decimals	18
totalSupply	The initial supply is 966666666 (tokens can be minted, can be destroyed, and the cap of minting is 1 billion)
Token type	ERC20

Table 1 The basic information of AIOZ token

According to the contract code logic, the initial token distribution is as follows:

Address	**Amount of tokens allocated**	**Release type**
(public sales) 0x076592ad72b79bBaBDD05aDd7d367f44f2CFf658	10333333	Immediately issued
(private sales) 0xF8477220f8375968E38a3B79ECA4343822b53af2	73000000	Initial issuance 25%, the remaining 75% will be released after 30 days, the release period is 30 days, each release 25%, and 3 times can be released
(team) 0x82E83054CC631C0Da85Ca67087E45ca31b93F29b	250000000	The tokens are all locked and will be released after 180 days. The release period is 30 days, each release 8%, and 13 times can be released.
(advisors) 0xBbf78c2Ee1794229e31af81c83F4d5125F08FE0F	50000000	The tokens are all locked and will be released after 90 days. The release period is 30 days, each release 8%, and 13 times can be released.
(marketing) 0x9E2F8e278585CAfD3308E894d2E09ffEc520b1E9	30000000	Initial issuance 10%, and the remaining 90% will be released after 30 days. The release period is 30 days, each release 5%, and 18 times can be released.
(exchange liquidity provision) 0x6c3D8872002B66C808aE462Db314B87962DCC7aF	23333333	Immediately issued
(ecosystem growth) 0xCFd6736a11e76c0e3418FEEbb788822211d92F1e	530000000	Immediately issued after 90 days

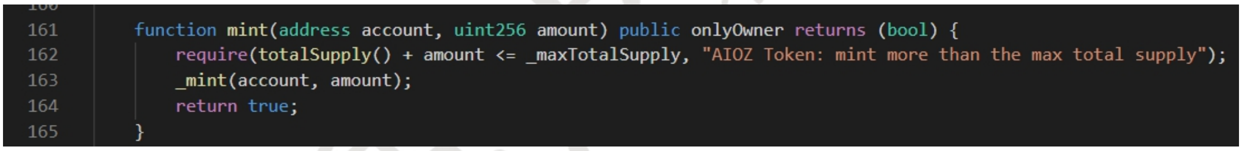
Table 2 AIOZ's initial token distribution

In addition, it should be noted that some of the beneficiary addresses in Table 2 cannot directly obtain the allocated tokens, but are stored in the corresponding time lock contract first, and must be released after the specified time.

- Result: Pass

3.1.3 Mint tokens

- Description: The manager of the contract can call the *mint* function to mint tokens to the specified address. As shown in the figure below, this function requires that the total amount of tokens after this minting cannot exceed the cap '_maxTotalSupply'.



```
    function mint(address account, uint256 amount) public onlyOwner returns (bool) {
        require(totalSupply() + amount <= _maxTotalSupply, "AIOZ Token: mint more than the max total supply");
        _mint(account, amount);
        return true;
    }
```

Figure 1 The source code of function mint

- Related functions: *mint, _mint*
- Safety Suggestion: None
- Result: Pass

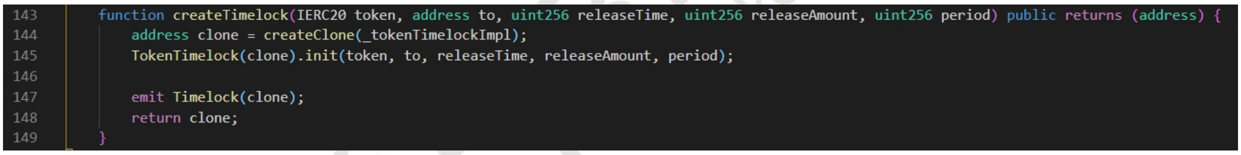
3.1.4 Burn tokens

- Description: The manager of the contract can call the *burn* function to destroy the tokens.
- Related functions: *burn, _burn*
- Safety Suggestion: None
- Result: Pass

3.2 The TokenTimelock Contract Audit

3.2.1 Create the time lock contract

- Description: Any user can call the *createTimelock* function to create a corresponding time lock contract. As shown in the figure below, this function uses the contract on the _tokenTimelockImpl address as a template to create a contract; and calls the *init* function on the new contract to initialize the time lock data.



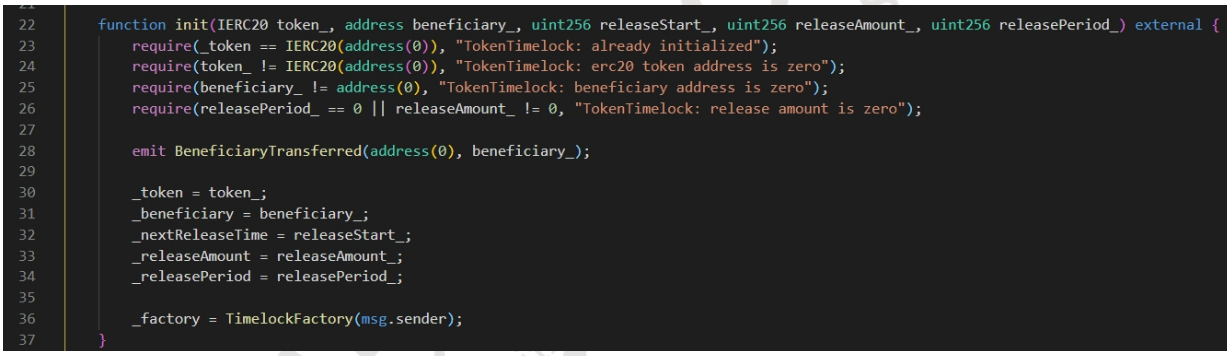
```
    function createTimelock(IERC20 token, address to, uint256 releaseTime, uint256 releaseAmount, uint256 period) public returns (address) {
        address clone = createClone(_tokenTimelockImpl);
        TokenTimelock(clone).init(token, to, releaseTime, releaseAmount, period);

        emit Timelock(clone);
        return clone;
    }
```

Figure 2 The source code of function createTimelock

- Related functions: *createTimelock, createClone, init*
- Result: Pass

3.2.2 Initialization of time lock contract

- Description: The user (under normal circumstances, the caller is the TimelockFactory contract) can call the *init* function of the time lock contract to initialize the time lock data. As shown in the figure below, the function requires the current token address _token must be the zero address, which can prevent this function from being called repeatedly.



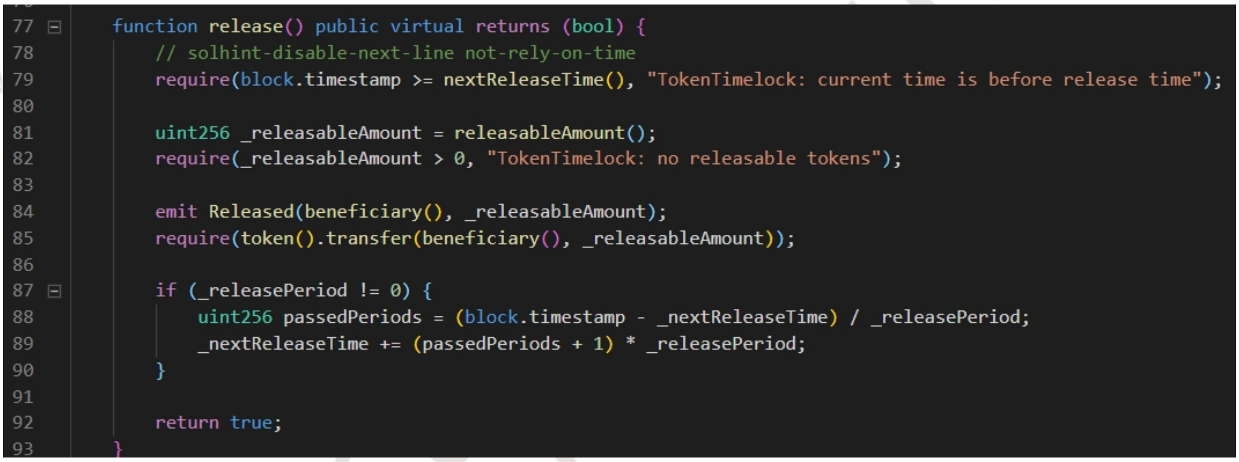
```
function init(IERC20 token_, address beneficiary_, uint256 releaseStart_, uint256 releaseAmount_, uint256 releasePeriod_) external {
    require(_token == IERC20(address(0)), "TokenTimelock: already initialized");
    require(token_ != IERC20(address(0)), "TokenTimelock: erc20 token address is zero");
    require(beneficiary_ != address(0), "TokenTimelock: beneficiary address is zero");
    require(releasePeriod_ == 0 || releaseAmount_ != 0, "TokenTimelock: release amount is zero");

    emit BeneficiaryTransferred(address(0), beneficiary_);

    _token = token_;
    _beneficiary = beneficiary_;
    _nextReleaseTime = releaseStart_;
    _releaseAmount = releaseAmount_;
    _releasePeriod = releasePeriod_;

    _factory = TimelockFactory(msg.sender);
}
```

Figure 3 The source code of function init

- Related functions: *init*
- Safety Suggestion: None
- Result: Pass

3.2.3 Release locked tokens

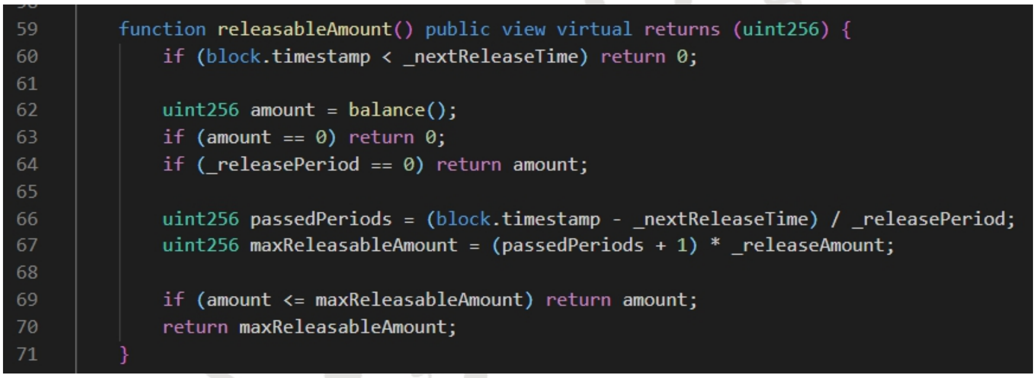
- Description: The user can call the *release* function of the contract to release the currently releasable locked tokens. As shown in the figure below, the function requires that the current timestamp must not be less than the next release timestamp, and the number of tokens that can be released is greater than 0.



```
function release() public virtual returns (bool) {
    // solhint-disable-next-line not-rely-on-time
    require(block.timestamp >= nextReleaseTime(), "TokenTimelock: current time is before release time");

    uint256 _releasableAmount = releasableAmount();
    require(_releasableAmount > 0, "TokenTimelock: no releasable tokens");

    emit Released(beneficiary(), _releasableAmount);
    require(token().transfer(beneficiary(), _releasableAmount));

    if (_releasePeriod != 0) {
        uint256 passedPeriods = (block.timestamp - _nextReleaseTime) / _releasePeriod;
        _nextReleaseTime += (passedPeriods + 1) * _releasePeriod;
    }

    return true;
}
```

Figure 4 The source code of function release

The *releasableAmount* function is used to calculate the amount of tokens that can be released under the current timestamp. As shown in the figure below, if the current timestamp is less than the timestamp of the next release or the contract balance is 0, the releasable amount is directly returned as 0; if the release period _releasePeriod is 0, the contract balance is directly returned; otherwise, the function will be based on the current time and the timestamp of the next release to calculate the amount of tokens that can be released.



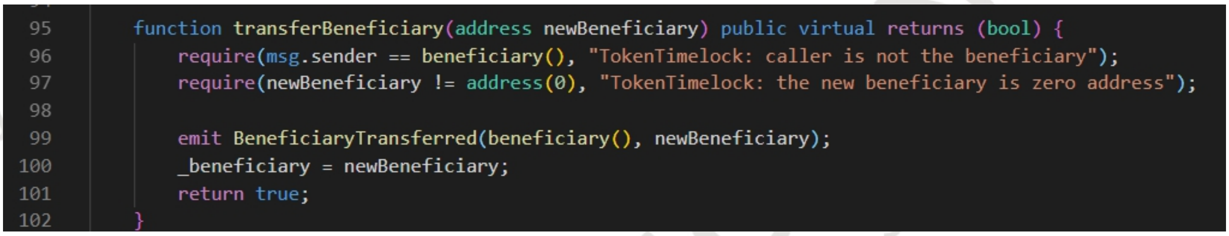
```
    function releasableAmount() public view virtual returns (uint256) {
        if (block.timestamp < _nextReleaseTime) return 0;

        uint256 amount = balance();
        if (amount == 0) return 0;
        if (_releasePeriod == 0) return amount;

        uint256 passedPeriods = (block.timestamp - _nextReleaseTime) / _releasePeriod;
        uint256 maxReleasableAmount = (passedPeriods + 1) * _releaseAmount;

        if (amount <= maxReleasableAmount) return amount;
        return maxReleasableAmount;
    }
```

Figure 5 The source code of function releasableAmount

- Related functions: *release, releasableAmount*
- Safety Suggestion: None
- Result: Pass

3.2.4 Update the beneficiary of locked tokens

- Description: The beneficiary address of the locked tokens can call the *transferBeneficiary* function to update the beneficiary address



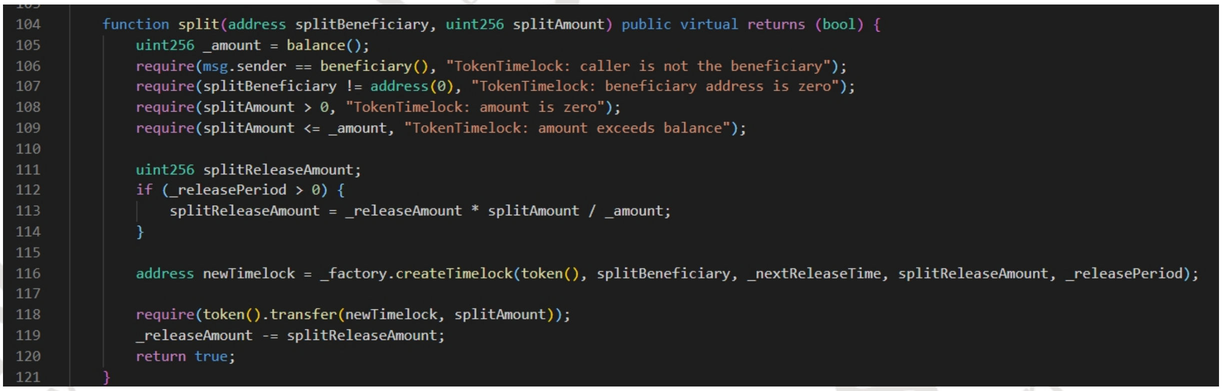
```
    function transferBeneficiary(address newBeneficiary) public virtual returns (bool) {
        require(msg.sender == beneficiary(), "TokenTimelock: caller is not the beneficiary");
        require(newBeneficiary != address(0), "TokenTimelock: the new beneficiary is zero address");

        emit BeneficiaryTransferred(beneficiary(), newBeneficiary);
        _beneficiary = newBeneficiary;
        return true;
    }
```

Figure 6 The source code of function transferBeneficiary

- Related functions: *transferBeneficiary*
- Safety Suggestion: None
- Result: Pass

3.2.5 Split locked tokens

- Description: The beneficiary address of the locked tokens can call the *split* function to split the unreleased tokens into a new time lock contract by a specified amount, and calculate the lock data after the split for initialization.



```
    function split(address splitBeneficiary, uint256 splitAmount) public virtual returns (bool) {
        uint256 _amount = balance();
        require(msg.sender == beneficiary(), "TokenTimelock: caller is not the beneficiary");
        require(splitBeneficiary != address(0), "TokenTimelock: beneficiary address is zero");
        require(splitAmount > 0, "TokenTimelock: amount is zero");
        require(splitAmount <= _amount, "TokenTimelock: amount exceeds balance");

        uint256 splitReleaseAmount;
        if (_releasePeriod > 0) {
            splitReleaseAmount = _releaseAmount * splitAmount / _amount;
        }

        address newTimelock = _factory.createTimelock(token(), splitBeneficiary, _nextReleaseTime, splitReleaseAmount, _releasePeriod);

        require(token().transfer(newTimelock, splitAmount));
        _releaseAmount -= splitReleaseAmount;
        return true;
    }
```

Figure 7 The source code of function split

As shown in Figure 7, the function will calculate the number of each release of the new time lock contract according to the ratio of the specified number of split to the number of remaining tokens in this contract, and other locking data will be consistent with this contract; then send this part of the lock tokens to the new time lock contract; finally update the _releaseAmount of this contract.

- Related functions: *split, createTimelock*
- Safety Suggestion: None
- Result: Pass

4. Conclusion

Beosin(ChengduLianAn) conducted a detailed audit on the design and code implementation of the smart contracts contract AIOZ. All the issues found during the audit have been written into this audit report. The overall audit result of the smart contract AIOZ is **Pass**.

Beosin

BEOSIN

Blockchain Security

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com