Smart contract audit report

NAMA FINANCE

CHAINLION

NO. OC002209080002

September 8, 2022

CHAINLION

CATALOGUE

1. PROJECT SUMMARY

Entry type	Specific description
Entry name	NAMA Finance
Project type	ERC-20/NFT
Application platform	ETH/BSC/Polygon/Avalanche

2. AUDIT SUMMARY

Entry type	Specific description
Project cycle	September/05/2022-September/08/2022
Audit method	Black box test、White box test、Grey box test
Auditors	THREE

3. VULNERABILITY SUMMARY

Audit results are as follows:

Entry type	Specific description
Serious vulnerability	0
High risk vulnerability	0
Moderate risk	0
Low risk vulnerability	0

Security vulnerability rating description:

1) **Serious vulnerability:** Security vulnerabilities that can directly cause token contracts or user capital losses, For example: shaping overflow vulnerability、Fake recharge vulnerability、Reentry attacks, vulnerabilities, etc.

2) **High risk vulnerability:** Security vulnerabilities that can directly cause the contract to fail to work normally, such as reconstructed smart contract caused by constructor design error, denial of service vulnerability caused by unreasonable design of require / assert detection conditions, etc.

3) **Moderate risk:** Security problems caused by unreasonable business logic design, such as accuracy problems caused by unreasonable numerical operation sequence design, variable ambiguous naming, variable coverage, call injection, conditional competition, etc.

4) **Low risk vulnerability:** Security vulnerabilities that can only be triggered by users with special permissions, such as contract backdoor vulnerability, duplicate name pool addition vulnerability, non-standard contract coding, contract detection bypass, lack of necessary events for key state variable change, and security vulnerabilities that are harmful in theory but have harsh utilization conditions.

4. EXECUTIVE SUMMARY

This report is prepared for **NAMA Finance** smart contract, The purpose is to find the security vulnerabilities and non-standard coding problems in the smart contract through the security audit of the source code of the smart contract. This audit mainly involves the following test methods:

White box test

Conduct security audit on the source code of smart contract and check the security issues such as coding specification, DASP top 10 and business logic design

Grey box test

Deploy smart contracts locally and conduct fuzzy testing to check function robustness, function call permission and business logic security

Black box test

Conduct security test attacks on smart contracts from the perspective of attackers, combined with black-and-white and testing techniques, to check whether there are exploitable vulnerabilities.

This audit report is subject to the latest contract code provided by

the current project party, does not include the newly added business logic function module after the contract upgrade, does not include new attack methods in the future, and does not include web front-end security and server-side security.

5. Directory structure

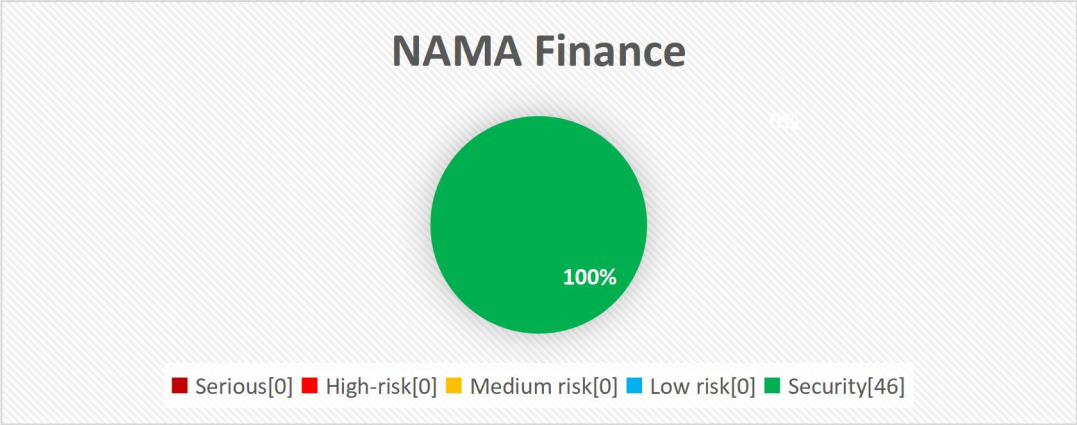
```
| BaseLoan.sol
|   FundingPool.sol
|   LoanRegistry.sol
|   PeerLoan.sol
|   TokenVault.sol
├──interfaces
|       DataTypes.sol
|       IBaseLoan.sol
|       ILiquidation.sol
|       IStakingRewards.sol
|       RewardsDistributionRecipient.sol
├──token
|       NamaToken.sol
└──utils
        Owned.sol
        StringAddressUtils.sol
```

6. File hashes

Contract	SHA1 Checksum
BaseLoan.sol	836E6AC290F7BA991B892C0B172F6953A3E7C600
FundingPool.sol	0E5DBDF29D31645E4894AB69E0EE2760C8D16788

LoanRegistry.sol	341A471C5DC1469B4333EA0817ACDA2592813BA5
PeerLoan.sol	434020F54453D3B2E62B28022B76AB30DEE987E0
TokenVault.sol	2C3DE597EB0F2D757FD0C24042D2EEB3D0060520
NamaToken.sol	FD7A5D7D2744E997ACDA3109D043B434D8D71870
Owned.sol	0293F196927C6C5D6172F16B6309595450E54EC6
StringAddressUtils.sol	A638915780F6221EC51E474C6102DCC348A9CD13

7. Vulnerability distribution



8. Audit content

8.1. Coding specification

Smart contract supports contract development in programming languages such as solid, Vyper, C + +, Python and rust. Each programming language has its own coding specification. In the development process, the coding specification of the development language should be strictly followed to avoid security problems such as business function design defects.

8.1.1. Compiler Version 【security】

Audit description： The compiler version should be specified in the smart contract code. At the same time, it is recommended to use the latest compiler version. The old version of the compiler may cause various known security problems. At present, the latest version is v 0.8 x. And this version has been protected against shaping overflow.

Audit results： According to the audit, the compiler version used in the smart contract code is 0.8.10, so there is no such security problem.

Safety advice：NONE.

8.1.2. Return value verification 【security】

Audit description： Smart contract requires contract developers to strictly follow EIP / tip and other standards and specifications during contract development. For transfer, transferfrom and approve functions, Boolean values should be returned to feed back the final execution results. In the smart contract, the relevant business logic code often calls the transfer or transferfrom function to transfer. In this case, the return value involved in the transfer operation should be strictly checked to determine whether the transfer is successful or not, so as to avoid security vulnerabilities such as false recharge caused by the lack of return value verification.

Audit results： According to the audit, there is no embedded function calling the official standards transfer and transferfrom in the smart contract, so there is no such security problem.

Safety advice：NONE.

8.1.3. Constructor writing 【security】

Audit description： In solid v0 The smart contract written by solidity before version 4.22 requires that the constructor must be consistent with the contract name. When the constructor name is inconsistent with the contract name, the constructor

will become an ordinary public function. Any user can call the constructor to initialize the contract. After version V 0.4.22, The constructor name can be replaced by constructor, so as to avoid the coding problems caused by constructor writing.

Audit results： After audit, the constructor in the smart contract is written correctly, and there is no such security problem.

```
11  /// 1e9 * 1e18 1B max
12  /// @notice This is the NAMA token contract
13  contract NamaToken is ERC20, ERC20Pausable, ERC20Burnable, ERC20Snapshot, Ownable {
14      error ZeroAddress();
15      error NotAllowed();
16
17      mapping(address => bool) public minters;
18
19      event AddMinter(address indexed minter);
20      event RemoveMinter(address indexed minter);
21
22      constructor() ERC20("NAMA Token", "NAMA") {
23          _mint(_msgSender(), 1 * 10 ** decimals());
24      }
25
```

Safety advice：NONE.

8.1.4. Key event trigger【security】

Audit description： Most of the key global variable initialization or update operations similar to setXXX exist in the smart contract. It is recommended to trigger the corresponding event through emit when operating on similar key events.

Audit results： After audit, there is no such security problem.

```
34      function setPause(bool _flag) external onlyOwner {
35          if (_flag) {
36              _pause();
37          } else {
38              _unpause();
39          }
40      }
```

Safety advice：NONE.

8.1.5. Address non-zero check【security】

Audit description： The smart contract initializes the key information of the contract through the constructor. When it comes to address initialization, the address should be non-zero checked to avoid irreparable economic losses.

Audit results： After audit, there is no such security problem.

```
130
131     /// @dev Updates the peer loan contract address
132     function setPeerLoan(address _peerAddr) external onlyOwner {
133         peerLoan = _peerAddr;
134     }
135
136     /// @dev Updates the pool loan contract address
137     function setPoolLoan(address _poolAddr) external onlyOwner {
138         poolLoan = _poolAddr;
139     }
```

Safety advice： **NONE.**

8.1.6. Code redundancy check【security】

Audit description： The deployment and execution of smart contracts need to consume certain gas costs. The business logic design should be optimized as much as possible, while avoiding unnecessary redundant code to improve efficiency and save costs.

Audit results： After audit, there is no such security problem.

Safety advice： **NONE.**

8.2. Coding design

DASP top 10 summarizes the common security vulnerabilities of smart contracts.

Smart contract developers can study smart contract security vulnerabilities before developing contracts to avoid security vulnerabilities during contract development. Contract auditors can quickly audit and check the existing security vulnerabilities of smart contracts according to DASP top 10.

8.2.1. Shaping overflow detection 【security】

Audit description： Solid can handle 256 digits at most. When the number is unsigned, the maximum value will overflow by 1 to get 0, and 0 minus 1 will overflow to get the maximum value. The problem of shaping overflow often appears in the relevant logic code design function modules such as transaction transfer, reward calculation and expense calculation. The security problems caused by shaping overflow are also very serious, such as excessive coinage, high sales and low income, excessive distribution, etc. the problem of shaping overflow can be solved by using solid V 0.8 X version or by using the safemath library officially provided by openzenppelin.

Audit results： According to the audit, the smart contract is applicable to the compiler of version 0.8.0, and the safemath library is used for numerical operation, which better prevents the problem of shaping overflow.

```
//SPDX-License-Identifier: BUSL-1.1

pragma solidity 0.8.9;

import {ERC20} from "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import {ERC20Burnable} from "@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol";
import {ERC20Pausable} from "@openzeppelin/contracts/token/ERC20/extensions/ERC20Pausable.sol";
import {ERC20Snapshot} from "@openzeppelin/contracts/token/ERC20/extensions/ERC20Snapshot.sol";
import {Ownable} from "@openzeppelin/contracts/access/Ownable.sol";

/// 1e9 * 1e18 1B max
/// @notice This is the NAMA token contract
```

Safety advice：NONE.

8.2.2. Reentry detection【security】

Audit description： The in solidity provides call Value(), send(), transfer() and other functions are used for transfer operation. When call When value() sends ether, it will send all gas for transfer operation by default. If the transfer function can be called recursively again through call transfer, it can cause reentry attack.

Audit results： After audit, there is no such security problem.

Safety advice：NONE.

8.2.3. Rearrangement attack detection【security】

Audit description：Rearrangement attack means that miners or other parties try to compete with smart contract participants by inserting their information into the list or mapping, so that attackers have the opportunity to store their information in the contract.

Audit results： After audit, there is no such security problem.

Safety advice：NONE.

8.2.4. Replay Attack Detection【security】

Audit description： When the contract involves the business logic of delegated management, attention should be paid to the non reusability of verification to avoid replay attacks. In common asset management systems, there are often delegated management businesses. The principal gives the assets to the trustee for management, and the principal pays a certain fee to the trustee. In similar delegated management scenarios, it is necessary to ensure that the verification information will become invalid once used.

Audit results： After audit, there is no such security problem.

Safety advice：NONE.

8.2.5. False recharge detection【security】

Audit description：When a smart contract uses the transfer function for transfer, it should use require / assert to strictly check the transfer conditions. It is not recommended to use if Use mild judgment methods such as else to check, otherwise it will misjudge the success of the transaction, resulting in the security problem of false recharge.

Audit results： After audit, there is no such security problem.

Safety advice：NONE.

8.2.6. Access control detection 【security】

Audit description： Solid provides four function access domain Keywords: public, private, external and internal to limit the scope of function. In the smart contract, the scope of function should be reasonably designed to avoid the security risk of improper access control. The main differences of the above four keywords are as follows:

1．public： The marked function or variable can be called or obtained by any account, which can be a function in the contract, an external user or inherit the function in the contract

2．external： The marked functions can only be accessed from the outside and cannot be called directly by the functions in the contract, but this can be used Func() calls this function as an external call

3．private： Marked functions or variables can only be used in this contract (Note: the limitation here is only at the code level. Ethereum is a public chain, and anyone can directly obtain the contract status information from the chain)

4．internal: It is generally used in contract inheritance. The parent contract is marked as an internal state variable or function, which can be directly accessed and called by the child contract (it cannot be directly obtained and called externally)

Audit results： After audit, there is no such security problem.

Safety advice：NONE.

8.2.7. Denial of service detection 【security】

Audit description：Denial of service attack is a DoS attack on Ethereum contract, which makes ether or gas consume a lot. In more serious cases, it can make the contract code logic unable to operate normally. The common causes of DoS attack are: unreasonable design of require check condition, uncontrollable number of for cycles, defects in business logic design, etc.

Audit results： After audit, there is no such security problem.

Safety advice：NONE.

8.2.8. Conditional competition detection 【security】

Audit description： The Ethereum node gathers transactions and forms them into blocks. Once the miners solve the consensus problem, these transactions are considered effective. The miners who solve the block will also choose which transactions from the mine pool will be included in the block. This is usually determined by gasprice transactions. Attackers can observe whether there are

transactions in the transaction pool that may contain problem solutions, After that, the attacker can obtain data from this transaction, create a higher-level transaction gasprice, and include its transaction in a block before the original, so as to seize the original solution.

Audit results： After audit, there is no such security problem.

Safety advice：NONE.

8.2.9. Consistency detection 【security】

Audit description： The update logic in smart contract (such as token quantity update, authorized transfer quota update, etc.) is often accompanied by the check logic of the operation object (such as anti overflow check, authorized transfer quota check, etc.), and when the update object is inconsistent with the check object, the check operation may be invalid, Thus, the conditional check logic is ignored and unexpected logic is executed. For example, the authorized transfer function function function transfer from (address _from, address _to, uint256 _value) returns (bool success) is used to authorize others to transfer on behalf of others. During transfer, the permission [_from] [MSG. Sender] authorized transfer limit will be checked, After passing the check, the authorized transfer limit will be updated at the same time of

transfer. When the update object in the update logic is inconsistent with the check object in the check logic, the authorized transfer limit of the authorized transfer user will not change, resulting in that the authorized transfer user can transfer all the assets of the authorized account.

Audit results： After audit, there is no such security problem.

Safety advice：NONE.

8.2.10. Variable coverage detection【security】

Audit description： Smart contracts allow inheritance relationships, in which the child contract inherits all the methods and variables of the parent contract. If a global variable with the same name as the parent contract is defined in the child contract, it may lead to variable coverage and corresponding asset losses.

Audit results： After audit, there is no such security problem.

Safety advice：NONE.

8.2.11. Random number detection【security】

Audit description： Random numbers are often used in smart contracts. When designing the random number generation function, the generation and selection of

random seeds should avoid the data information that can be queried on the blockchain, such as block Number and block Timestamp et al. These data are vulnerable to the influence of miners, resulting in the predictability of random numbers to a certain extent.

Audit results： After audit, there is no such security problem.

Safety advice：NONE.

8.2.12. Numerical operation detection 【security】

Audit description： Solidity supports addition, subtraction, multiplication, division and other conventional numerical operations, but solidty does not support floating-point types. When multiplication and division operations exist at the same time, the numerical operation order should be adjusted reasonably to reduce the error as much as possible.

Audit results： After audit, there is no such security problem.

Safety advice：NONE.

8.2.13. Call injection detection 【security】

Audit description： In the solid language, you can call a contract or a method of a

local contract through the call method. There are roughly two ways to call: < address > Call (method selector, arg1, arg2,...) or < address > Call (bytes). When using call call, we can pass method selectors and parameters by passing parameters, or directly pass in a byte array. Based on this function, it is recommended that strict permission check or hard code the function called by call when using call function call.

Audit results： After audit, there is no such security problem.

Safety advice：NONE.

8.3. Business logic

Business logic design is the core of smart contract. When using programming language to develop contract business logic functions, developers should fully consider all aspects of the corresponding business, such as parameter legitimacy check, business permission design, business execution conditions, interaction design between businesses, etc.

8.3.1. Constructor initialization logic 【security】

Audit description： Conduct security audit on the constructor initialization and business logic design in the contract, and check whether the initialization value is consistent with the requirements document.

Audit results： The constructor initialization business logic design in the contract

is correct, and no relevant security risks are found.

Code file：NamaToken.sol 22~24

Code information：

```
/// 1e9 * 1e18 1B max
/// @notice This is the NAMA token contract
contract NamaToken is ERC20, ERC20Pausable, ERC20Burnable, ERC20Snapshot, Ownable {
    error ZeroAddress();    //Zero address error
    error NotAllowed();     //not allow

    mapping(address => bool) public minters;    //miner Mapping

    event AddMinter(address indexed minter);
    event RemoveMinter(address indexed minter);

    constructor() ERC20("NAMA Token", "NAMA") {    //Specify token name、Sysmbol
        _mint(_msgSender(), 1 * 10 ** decimals());     //Total amount of tokens issued during initialization
```

Safety advice：NONE.

8.3.2. Logical design of burn token destruction【security】

Audit description： Conduct security audit on the logical design of the contract burn token destruction business to check whether there are related business security problems such as shaping overflow and improper authority control.

Audit results： After audit, the burn token destruction logic was designed and repaired.

Code file： NamaToken.sol 26~28

Code information：

```
    function burn(address _addr, uint256 _amount) external onlyOwner {
        _burn(_addr, _amount);
```

```
}
```

Safety advice：NONE.

8.3.3. Logic design of mint token issuance 【security】

Audit results： Conduct security audit on the logic design of the contract Mint token issuance business to check whether there are related business security problems such as shaping overflow and improper authority control.

Audit results： After audit, the logic design of mint token issuance has been repaired.

Code file： NamaToken.sol 42~48

Code information：

```
function mint(address _to, uint256 _amount) whenNotPaused external {
    if (_msgSender() != owner() && !minters[_msgSender()]) {        //When the caller is not the owner or non miner of the contract, it is not allowed to call
        revert NotAllowed();
    }

    _mint(_to, _amount);
}
```

Safety advice：NONE.

8.3.4. Addminter business logic design 【security】

Audit description： Conduct security audit on the business logic design of the contract addminter add miner to check whether there are security problems such as improper authority control and design defects.

Audit results： Addminter in the contract adds the miner's business logic, and

the design is correct.

Code file： NamaToken.sol 50~57

Code information：

```
    function addMinter(address _addr) whenNotPaused external onlyOwner { //Only the owner of the contract is allowed to call when when notpaused is not false
        if (_addr == address(0)) { //Address non-zero check
            revert ZeroAddress();
        }

        minters[_addr] = true; //Update mapping and set the address as the miner's address
        emit AddMinter(_addr);
    }
```

Safety advice： NONE.

8.3.5. Removeminter business logic design 【security】

Audit description： Conduct security audit on the business logic design of the contract removeminter removal miner to check whether there are security problems such as improper permission control and design defects.

Audit results： The removeminter in the contract removes the miner, and the business logic design is correct.

Code file： NamaToken.sol 59~66

Code information：

```
    function removeMinter(address _addr) whenNotPaused external onlyOwner {//Only the owner of the contract is allowed to call when when notpaused is not false
        if (_addr == address(0)) {//Address non-zero check
            revert ZeroAddress();
        }
```

```
        delete minters[_addr];//Update mapping and remove the current address
        emit RemoveMinter(_addr);
        }
```

Safety advice：NONE.

8.3.6. Logic design of nominateneowner【security】

Audit description： Conduct security audit on the business logic design of the new owner nominated by the nominatenewner to check whether there are security problems such as improper authority control and design defects.

Audit results： The logic design of nominateNewOwner is not fixed.

Code file： Owned.sol 18~28

Code information：

```
        function nominateNewOwner(address _owner) external onlyOwner { //Nominate a new owner, only the contract owner is allowed to call
            nominatedOwner = _owner;
            emit OwnerNominated(_owner);
    }
    function acceptOwnership() external {
            require(_msgSender() == nominatedOwner, "You must be nominated before you can accept ownership");
            emit OwnerChanged(owner, nominatedOwner);
            owner = nominatedOwner;
            nominatedOwner = address(0);
    }
```

Safety advice：NONE.

8.3.7. Liquidate loan logic 【security】

Audit description： Conduct security audit on the business logic design of the loan liquidation liquidateloan in the contract to check whether there are security problems such as improper authority control and design defects.

Audit results： The business logic design of loan liquidation (liquidateloan) in the contract is correct.

Code file： BaseLoad.sol 56~72

Code information：

```
function liquidateLoan(bytes32 _loanId) external { //Clearing loans
        LoanMetadata storage loan = loans[_loanId];    //According to_ Loanid get loan information
        if (liquidationAddress == address(0)) { //Check whether the clearing address is empty (it is recommended to use require() to check)
            revert ZeroAddress(); //Zero address error
        }
        if (block.timestamp <= (loan.withdrawAt + loan.term.duration + gracePeriod)) { //Check whether it is within the loan cycle. If it is, liquidation is not allowed
            revert NotAllowed();
        }

        loan.status = LoanStatus.DEFAULTED; //Update the loan status to "defaulted"

        ILoanRegistry(loanRegistry).releaseAsset(_loanId, liquidationAddress); //Release assets

        ILiquidation(liquidationAddress).liquidate(_loanId); //Clearing loans

        emit Liquidated(_loanId, _msgSender());
    }
    /// @dev Only the PeerLoan and PoolLoan can call the function
    modifier onlyNamaLoans() {
        if (_msgSender() != peerLoan && _msgSender() != poolLoan) {
            revert NotAllowed();
```

```
        }
        _;
    }
    /// @notice Releases the assets locked in the loan `_loanId`
    function releaseAsset(bytes32 _loanId, address _to) external onlyNamaLoans { //Release lock on loan`_ Assets in loanid`
        bool succeed = TokenVault(loanVaults[_loanId]).releaseAsset(_loanId, _to);
        if (!succeed) {
            revert ErrorToReleaseAsset();
        }
    }
    constructor(address _vaultOwner) Owned(_vaultOwner) {
        loanManager = _msgSender();
    }

    modifier onlyLoanManager() {
        if (_msgSender() != loanManager) {
            revert Unauthorised();
        }

        _;
    }
    /// @notice Releases the secured NFTs in the loan to the borrower when it's get repaid/cancelled,
    ///     or to the liquidation contract when it's get defaulted
    /// @notice This can only be called by the NAMA loan contracts and only if the loanId is still
    /// @param _loanId The NFT contract address
    /// @param _to The recipient, if it's zero address then transfer the NFTs to the owner of this vault.
    function releaseAsset(bytes32 _loanId, address _to)
        external
        onlyLoanManager
        returns (bool)
    {
        _transferNFTs(_loanId, _to == address(0) ? owner : _to); //transfer accounts

        delete securedAssetsInTheLoan[loans[_loanId]]; //Remove fixed assets
        delete loans[_loanId]; //Delete loan_ loanId
```

```
            return true;
        }
        /// @notice Transfers the NFTs to `_to`
        /// @param _loanId The loan id
        /// @param _to The recipient
        function _transferNFTs(bytes32 _loanId, address _to) private returns (bool) {
            DataTypes.CollateralizedAsset[] memory _assets = securedAssetsInTheLoan[loans[_loanId]];
//Acquisition of mortgaged assets
            uint256 size = _assets.length;
            for (uint256 i = 0; i < size; i++) {
                address _nftAddr = _assets[i].assetAddress;
                uint256[] memory assetIds = _assets[i].assetIds;

                uint256 tokenSize = assetIds.length;
                for (uint256 j = 0; j < tokenSize; j++) {
                    IERC721(_nftAddr).safeTransferFrom(address(this),    _to,    assetIds[j]);    //Asset
liquidation
                }
            }

            return true;
        }
}
```

Safety advice：NONE.

8.3.8. Logical design of withdrawnativetoken【security】

Audit description： Conduct security audit on the business logic design of withdrawnativetoken in the contract to check whether there are security problems such as improper permission control and design defects.

Audit results： withdrawNativeToken logic design repair.

Code file： BaseLoad.sol 74~76

Code information：

```
function withdrawNativeToken(uint256 _amount) public onlyOwner {
    payable(_msgSender()).transfer(_amount == 0 ? address(this).balance : _amount);
}
```

Safety advice： ： NONE.

8.3.9. Logical design of loanexpirationtime 【security】

Audit description： Conduct security audit on the loanexpirationtime business logic design in the contract to check whether there are security problems such as improper permission control and design defects.

Audit results： The loanexpirationtime in the contract is designed correctly.

Code file： BaseLoad.sol 114~118

Code information：

```
function loanExpirationTime(bytes32 _loanId) external view returns (uint256) { //Get loan expiration time
    LoanMetadata memory loan = loans[_loanId]; //Retrieve loan
    return loan.term.duration + loan.withdrawAt; //Calculated from the withdrawal time, plus the period is the final maturity time
}
```

Safety advice： ： If there is no special function, please delete the function.

8.3.10. Totalowed logic design 【security】

Audit description： Conduct a security audit on the business logic design of the

total owed amount in the contract to check whether there are security problems such as improper authority control and design defects.

Audit results： The totalowed design in the contract is correct.

Code file： BaseLoad.sol 120~137

Code information：

```
function totalOwed(bytes32 _loanId) external view returns (uint256) {
        LoanMetadata memory loan = loans[_loanId];     //According to_ Loanid: retrieve loan information
        if (loan.withdrawAt == 0 || loan.status != LoanStatus.WITHDRAWN) return 0; //If the loan is not withdrawn, or the loan status is not withdrawn, 0 will be returned directly

        return loan.fundRaised + interestOwed(_loanId); //Loan principal + interest owed
    }
    function interestOwed(bytes32 _loanId) public view returns (uint256) {
        LoanMetadata memory loan = loans[_loanId]; //According to_ Loanid: retrieve loan information
        if (loan.withdrawAt == 0 || loan.status != LoanStatus.WITHDRAWN) return 0;//If the loan is not withdrawn, or the loan status is not withdrawn, 0 will be returned directly

        return _interestOwed(
            loan.fundRaised - loan.amountRepaid,
            loan.term.interestRate,
            block.timestamp - loan.withdrawAt,
            loan.term.duration
        ); //Calculate interest
    }
    function _interestOwed(
        uint256 _borrowAmount,
        uint256 _loanInterestRate,
        uint256 _loanDurationToNow,
        uint256 _loanExpirationTime
    )
        internal
```

```
        pure
        returns (uint256)
    {
        if (_loanDurationToNow > _loanExpirationTime) { //Check whether the maximum time limit is exceeded. If it is exceeded, set it as the expiration time
            _loanDurationToNow = _loanExpirationTime;
        }

        uint256 toDateInDays = _loanDurationToNow / 1 days; //How many days
        if (toDateInDays == 0) {
            toDateInDays = 1;
        } else if (_loanDurationToNow % 1 days != 0) {
            toDateInDays += 1;
        }

        uint256 interestPerDay = (_borrowAmount * _loanInterestRate) / SCALAR / 100 / 365; //Calculate daily interest
        return interestPerDay * toDateInDays;    //Calculate total interest
    }
```

Safety advice：NONE.

8.3.11. Eared business logic design【security】

Audit description： Conduct security audit on the eared business logic design in the contract to check whether there are security problems such as improper authority control and design defects.

Audit results： The eared business design in the contract is correct.

Code file： FundingPool.sol 102~104

Code information：

```
/// @dev Returns the total earned of an account
    function earned(address _account) public view returns (uint256) {
```

```
        return ((balanceOf[_account] * (rewardPerToken() - userRewardPerTokenPaid[_account])) / 1e18) + rewards[_account]; //Calculate the reward amount (net)
    }
    function rewardPerToken() public view returns (uint256) {
        if (totalSupply == 0) {     //Check whether totalsupply is 0. If it is zero, it will return rewardpertokenstored by default
            return rewardPerTokenStored;
        }
        return rewardPerTokenStored + (((block.timestamp - lastUpdateTime) * rewardRate * 1e18) / totalSupply);    //If it is not zero, the reward that each token can get will be calculated
    }
```

Safety advice： ： NONE.

8.3.12. Exit business logic design【security】

Audit description：Conduct security audit on the exit business logic design in the contract to check whether there are security problems such as improper authority control and design defects.

Audit results： The exit business design in the contract is correct.

Code file： FundingPool.sol 146~149

Code information：

```
 function exit() external {
        withdraw(balanceOf[_msgSender()], _msgSender(), _msgSender()); //Withdrawal of pledge amount
        getReward(); //Withdrawal reward
    }
    function withdraw(
        uint256 _assets,
        address _receiver,
        address _owner
    ) public override nonReentrant whenNotPaused updateReward(_msgSender()) returns (uint256 shares) { //Anti reentry, update reward
        shares = super.withdraw(_assets, _receiver, _owner);
```

```
    }
    function getReward() public nonReentrant updateReward(_msgSender()) { //Anti reentry, update reward
        uint256 reward = rewards[_msgSender()];
        if (reward > 0) {
            rewards[_msgSender()] = 0;
            rewardsToken.safeTransfer(_msgSender(), reward);
            emit RewardPaid(_msgSender(), reward);
        }
    }
```

Safety advice：NONE.

8.3.13. Payout business logic design【security】

Audit description：Conduct security audit on the payout business logic design in the contract to check whether there are security problems such as improper authority control and design defects.

Audit results：Layout business logic design and repair.

Code file： FundingPool.sol 152~156

Code information：

```
// TODO need to check the status of the loan before payout
    function payout(bytes32 /**_loanId**/, address _to, uint256 _amount) external onlyControllers returns (bool) { ///**_loanId**/Although it has been annotated, the previous bytes32 remains, which has coding defects
        ERC20(asset).safeTransfer(_to, _amount);
        emit PaidOut(_to, _amount);
        return true;
    }
```

Safety advice：NONE.

8.3.14. Getreward business logic design【security】

Audit description： Conduct security audit on the getreward business logic design in the contract to check whether there are security problems such as improper permission control and design defects.

Audit results： The getreward logic design in the contract is correct.

Code file： FundingPool.sol 165~172

Code information：

```
function getReward() public nonReentrant updateReward(_msgSender()) {
        uint256 reward = rewards[_msgSender()];
        if (reward > 0) {
            rewards[_msgSender()] = 0;
            rewardsToken.safeTransfer(_msgSender(), reward);
            emit RewardPaid(_msgSender(), reward);
        }
    }
```

Safety advice：NONE.

8.3.15. Logic design of notifyrewardamount【security】

Audit description： Conduct security audit on the notifyrewardamount business logic design in the contract to check whether there are security problems such as improper permission control and design defects.

Audit results： The logic design of notifyrewardamount in the contract is correct.

Code file： FundingPool.sol 186~205

Code information：

```
function notifyRewardAmount(uint256 reward) external override onlyRewardsDistribution updateReward(address(0)) {
        if (block.timestamp >= periodFinish) { //Check whether a cycle is completed
            rewardRate = reward / rewardsDuration;    //Calculate rewardrate
        } else {
```

```
            uint256 remaining = periodFinish - block.timestamp;    //Calculate remaining time
            uint256 leftover = remaining * rewardRate;    //Calculate the quantity to be completed
            rewardRate = (reward + leftover) / rewardsDuration; //calculation rewardRate
        }

        // Ensure the provided reward amount is not more than the balance in the contract.
        // This keeps the reward rate in the right range, preventing overflows due to
        // very high values of rewardRate in the earned and rewardsPerToken functions;
        // Reward + leftover must be less than 2^256 / 10^18 to avoid overflow.
        uint balance = rewardsToken.balanceOf(address(this)); //Get asset quantity
        require(rewardRate <= balance / rewardsDuration, "Provided reward too high"); //inspect
rewardRate

        lastUpdateTime = block.timestamp; //Update time:
        periodFinish = block.timestamp + rewardsDuration;    //Update next cycle end time
        emit RewardAdded(reward);
        }
```

Safety advice：NONE.

8.3.16. Setrewardsduration logic design【security】

Audit description： Conduct security audit on the business logic design of setrewardsduration in the contract to check whether there are security problems such as improper permission control and design defects.

Audit results：The logical design of setrewardsduration in the contract is correct.

Code file： FundingPool.sol 207~214

Code information：

```
function setRewardsDuration(uint256 _rewardsDuration) external onlyOwner {
        require(
            block.timestamp > periodFinish,
            "Previous rewards period must be complete before changing the duration for the new
period"
        );
```

```
        rewardsDuration = _rewardsDuration;
        emit RewardsDurationUpdated(rewardsDuration);
        }
```

Safety advice：NONE.

8.3.17. Logic design of recoverc20 【security】

Audit description： Conduct security audit on the business logic design of recoverc20 in the contract to check whether there are security problems such as improper authority control and design defects.

Audit results： The logic design of recoverc20 in the contract is correct.

Code file： FundingPool.sol 216~221

Code information：

```
    // Added to support recovering LP Rewards from other systems such as BAL to be distributed to holders
    function recoverERC20(address tokenAddress, uint256 tokenAmount) external onlyOwner { //only allow owner calls
        require(tokenAddress != address(asset), "Cannot withdraw the staking token"); //Withdrawal of pledged token is not allowed
        ERC20(tokenAddress).safeTransfer(owner, tokenAmount);
        emit Recovered(tokenAddress, tokenAmount);
    }
```

Safety advice：NONE.

8.3.18. Logic design of createloan 【security】

Audit description： Conduct security audit on the createloan business logic design in the contract to check whether there are security problems such as improper permission control and design defects.

Audit results： The logic design of createloan in the contract is correct.

Code file： LoanRegistry.sol 54~116

Code information：

```
    /// @notice Creates a loan
    function createLoan(
        bytes32 _loanId,
        LoanTerm memory _term,
        LoanType _loanType,
        CollateralizedAsset[] memory _assets,
        string calldata _destRecvChain,
        string calldata _recverAddress
    )
        external
        payable
        whenNotPaused
        nonReentrant
    { //Prevent reentry, only call when notpaused
        if (_loanId.length == 0 || _term.duration == 0 || _assets.length == 0
            || _term.loanAmount == 0 || _term.targetAmount < _term.loanAmount) { //Check whether the number of parameters is correct
            revert ArgumentException();
        }

        if (_loanType != LoanType.P2M && _loanType != LoanType.AUCTION && _loanType != LoanType.POOL) { //Check whether the loan type is in the list
            revert NotSupportedLoanException();
        }

        address vaultAddr = tokenVaults[_msgSender()]; //Mortgage address
        if (_loanType == LoanType.POOL) { //If the loan type is pool mode
            // validate whether the assets are in the allowlist
            bool valid = _validateAssets(_assets); //Verify whether the asset is valid (within the allowed list)
            if (!valid) {
                revert NoSupportedNFTsException(); //Throw exception information if invalid
```

```
            }

            TokenVault vault = vaultAddr == address(0) ? new TokenVault(_msgSender()) :
TokenVault(vaultAddr); //Get vaultaddr address (no loan has been made before)
            bool succeed = _depositNFTs(_assets, _msgSender(), address(vault)); //Mortgage NFT
            if (!succeed) {
                revert ErrorToDepositNFTs();
            }
            vault.addCollaterals(_loanId, _assets); //Increase collateral
            IBaseLoan(poolLoan).createLoan{value: msg.value}(
                _loanId,
                _msgSender(),
                _term,
                _loanType,
                _destRecvChain,
                _recverAddress
            ); //Create loan
            loanVaults[_loanId] = address(vault);
        } else {
            TokenVault vault = vaultAddr == address(0) ? new TokenVault(_msgSender()) :
TokenVault(vaultAddr);
            bool succeed = _depositNFTs(_assets, _msgSender(), address(vault));
            if (!succeed) {
                revert ErrorToDepositNFTs();
            }
            vault.addCollaterals(_loanId, _assets);//Increase collateral
            IBaseLoan(peerLoan).createLoan(
                _loanId,
                _msgSender(),
                _term,
                _loanType,
                _destRecvChain,
                _recverAddress
            );
            loanVaults[_loanId] = address(vault);
        }//Create loan
```

```
    }
    /// @notice Validates whether the deposited assets are in the allowed list
    function _validateAssets(CollateralizedAsset[] memory _assets) internal view returns (bool) {
        uint256 size = _assets.length;
        for (uint i = 0; i < size; i++) {
            if (allowedAssets[_assets[i].assetAddress]) {
                return true;
            }
        }
        return false;
    }
    /// @notice Deposits NFTs to the corresponding token vault
    function _depositNFTs(CollateralizedAsset[] memory _assets, address _sender, address _recipient)
internal returns (bool) {
        // make sure all NFTs must be approved before transferring
        uint256 size = _assets.length;
        uint256 i = 0;
        for (; i < size; i++) {
            address _nftAddr = _assets[i].assetAddress;
            uint256[] memory assetIds = _assets[i].assetIds;
            uint256 tokenSize = assetIds.length;

            for (uint256 j = 0; j < tokenSize; j++) {
                IERC721(_nftAddr).safeTransferFrom(_sender, _recipient, assetIds[j]);
            }
        }

        return true;
    }
    /// @notice Records the secured NFTs to the corresponding loan.
    function addCollaterals(bytes32 _loanId, DataTypes.CollateralizedAsset[] calldata _assets)
external onlyLoanManager {
        loans[_loanId] = securedAssetKey;

        uint256 i = 0;
        uint256 size = _assets.length;
```

```
            for (; i < size; i++) {
                DataTypes.CollateralizedAsset memory ca = DataTypes.CollateralizedAsset({
                    assetAddress: _assets[i].assetAddress,
                    assetIds: _assets[i].assetIds
                });
                securedAssetsInTheLoan[securedAssetKey++].push(ca);
            }
        }
        function createLoan(
            bytes32 loanId,
            address borrower,
            LoanTerm memory loanTerm,
            LoanType loanType,
            string calldata destRecvChain,
            string calldata recverAddress
    ) external payable;    //Apply for loan
```

Safety advice：NONE.

8.3.19. Createloan logic design 2【security】

Audit description： Conduct security audit on the createloan business logic design in the contract to check whether there are security problems such as improper permission control and design defects.

Audit results： The logic design of createloan in the contract is correct.

Code file： PeerLoan.sol 29~63

Code information：

```
        function createLoan(
            bytes32 _loanId,
            address _borrower,
            LoanTerm memory _loanTerm,
            LoanType _loanType,
            string calldata _destRecvChain,
```

```
        string calldata _recverAddress
    )
        external
        payable
        onlyCreator
    { //Only Creator calls are allowed
        if (lendingAssets[_loanTerm.fundType] == address(0)) { //Check whether the lending asset address corresponding to fundtype is empty
            revert InvalidFundType();
        }

        LoanMetadata storage loan = loans[_loanId]; //According to_ Loanid get loan details
        if (loan.appliedAt != 0) {//Whether the application event is 0. If it is not 0, it indicates that an application has been made
            revert LoanAlreadyExisted();
        }

        loan.loanId = _loanId;
        loan.term = _loanTerm;
        loan.term.duration = _loanTerm.duration * 1 days;
        loan.appliedAt = block.timestamp;
        loan.loanType = _loanType;
        loan.status = LoanStatus.AWAITING;
        loan.borrower = _borrower;
        loan.createdOnChain = StringsUpgradeable.toString(block.chainid);

        loan.destRecvChain = _destRecvChain;
        loan.recverAddress = _recverAddress;

        emit LoanCreated(loan.loanId, loan.borrower, lendingAssets[loan.term.fundType], loan.term.loanAmount);
    }
```

Safety advice：NONE.

8.3.20. Fund business logic design【security】

Audit description: Conduct security audit on the logic design of fund business in the contract to check whether there are security problems such as improper authority control and design defects.

Audit results: The logical design of fund in the contract is correct.

Code file: PeerLoan.sol 67~82

Code information:

```
/// @notice lend an amount of `_amount` to the loan `_loanId`,
    ///    only the lender of the loan can call this
    function fund(bytes32 _loanId, uint256 _amount) external payable nonReentrant whenNotPaused
{ //Anti reentry, can be called when notpaused
        LoanMetadata memory loan = loans[_loanId];      //According to_ Loanid: get loan
information
        if (loan.borrower == _msgSender() || loan.status != LoanStatus.AWAITING || _amount == 0)
{ //Check the borrower, loan status and loan quantity
            revert NotAllowed();
        }

        if    (IERC20Upgradeable(lendingAssets[loan.term.fundType]).balanceOf(_msgSender())    <
_amount) { //Check whether the assets are sufficient
            revert InsufficientBalance();
        }

        if (loan.loanType == LoanType.P2M) { //If it is p2m type, call_ P2moffer completes
borrowing
            _p2mOffer(_loanId, _amount, loan.term.fundType);
        } else { //Otherwise, call_ auctionOffer
            _auctionOffer(_loanId, _amount, loan.term.fundType);
        }
    }
    function _p2mOffer(bytes32 _loanId, uint256 _amount, bytes32 _fundType) internal {
        LoanMetadata storage loan = loans[_loanId]; //According to_ Loanid get loan information
        uint256 leftover = loan.term.loanAmount - loan.fundRaised; //Get the remaining quantity to
be lent
```

```
            if (_amount > leftover) {
                _amount = leftover;
            }

            loan.fundRaised += _amount; //Update total loan quantity
            lenders[_loanId][_msgSender()] += _amount; //Update loan information
            loanLenders[_loanId].push(_msgSender()); //Add Borrower(_loanId->borrwer)

            if (loan.fundRaised == loan.term.loanAmount) { //If the loan quantity is consistent with the raised quantity, the loan is completed and the status is updated
                loan.status = LoanStatus.FUNDED;
            }

            IERC20Upgradeable(lendingAssets[_fundType]).safeTransferFrom(_msgSender(), address(this), _amount); //lend

            emit MakeOffer(_loanId, _msgSender(), loan.status, lenders[_loanId][_msgSender()], loan.fundRaised);
        }
        function _auctionOffer(bytes32 _loanId, uint256 _amount, bytes32 _fundType) internal {
            LoanMetadata storage loan = loans[_loanId];//According to_ Loanid get loan information
            if ((loan.fundRaised == 0 && _amount < loan.term.loanAmount) || _amount <= loan.fundRaised) { //Check whether the raised funds are 0 and the assets are less than the borrowed assets or the borrowed assets are less than the raised assets
                revert InvalidLoanAmount();
            }

            // if offerred amount is greater than the borrowing amount then set the offer amount to target amount
            if (_amount > loan.term.targetAmount) { //If the parameter amount is greater than the loan amount, set the parameter amount to the loan amount
                _amount = loan.term.targetAmount;
            }

            if (loanTopBidder[_loanId] != address(0)) { //Check if the highest bidder is found
                address bidder = loanTopBidder[_loanId];
```

```
                uint256 bidAmount = lenders[_loanId][bidder]; //Lending quantity

                loanTopBidder[_loanId] = address(0);
                lenders[_loanId][bidder] = 0;
                loan.fundRaised -= bidAmount; //Update raised assets

                IERC20Upgradeable(lendingAssets[_fundType]).safeTransfer(bidder, bidAmount);
            }

            loan.fundRaised = _amount;
            lenders[_loanId][_msgSender()] = _amount;
            loanTopBidder[_loanId] = _msgSender();

            if (loan.term.targetAmount != 0 && loan.fundRaised == loan.term.targetAmount)
{ //Complete the raising
                loan.status = LoanStatus.FUNDED;
            }

            IERC20Upgradeable(lendingAssets[_fundType]).safeTransferFrom(_msgSender(),
address(this), _amount);

            emit MakeOffer(_loanId, _msgSender(), loan.status, lenders[_loanId][_msgSender()],
_amount);
        }
```

Safety advice：NONE.

8.3.21. Revokeoffer business logic design【security】

Audit description： Conduct security audit on the revokeoffer revocation asset business logic design in the contract to check whether there are security problems such as improper authority control and design defects.

Audit results： The revokeoffer in the Contract cancels the asset, and the business logic design is correct.

Code file： PeerLoan.sol 86~102

Code information：

```
/// @notice Revokes the offer to a loan `_loanId`,
    ///    only works P2M type of loan and only the lender of the loan can call this
    function revokeOffer(bytes32 _loanId, uint256 _amount) external nonReentrant {
        LoanMetadata storage loan = loans[_loanId];//According to_ Loanid: get loan information
        if (loan.loanType != LoanType.P2M || _amount == 0
            || _amount > lenders[_loanId][_msgSender()]
            || (loan.status != LoanStatus.AWAITING && loan.status != LoanStatus.FUNDED))
{ //Check the loan status and whether to allow cancellation
            revert NotAllowed();
        }

        loan.fundRaised -= _amount; //Update the number of assets to be raised
        lenders[_loanId][_msgSender()] -= _amount;    //Update the borrower's loan quantity

        if (loan.fundRaised < loan.term.targetAmount) { //If the raised assets are smaller than the
target assets, the update status is not waiting
            loan.status = LoanStatus.AWAITING;
        }
        IERC20Upgradeable(lendingAssets[loan.term.fundType]).safeTransfer(_msgSender(),
_amount); //Cancellation of some assets
        emit RevokeOffer(_loanId, _msgSender(), loan.status, _amount, loan.fundRaised);
    }
```

Safety advice：NONE.

8.3.22. Cancelloanapplication business logic design【security】

Audit description： Conduct security audit on the logic design of cancelloanapplication cancellation loan application business in the contract to check whether there are security problems such as improper authority control and design defects.

Audit results： The logic design of cancelloanapplication in the contract is correct.

Code file： PeerLoan.sol 106~135

Code information：

```
/// @notice Cancels the loan application,
    ///   only the borrower of the loan can call this
    function cancelLoanApplication(bytes32 _loanId) external nonReentrant {
        LoanMetadata storage loan = loans[_loanId];            //According to_ Loanid: get loan information
        if (loan.borrower != _msgSender() || loan.status != LoanStatus.AWAITING) { //Check the processing status of the caller and debit / credit application to see if it is allowed to cancel the application
            revert NotAllowed();
        }

        loan.status = LoanStatus.CANCELLED; //Update loan status

        if (loan.loanType == LoanType.P2M) { //Check whether it is p2m loan type
            address[] memory lenderList = loanLenders[_loanId]; //Get borrower list
            uint256 size = lenderList.length;
            for (uint256 i = 0; i < size; i++) {
                uint256 amount = lenders[_loanId][lenderList[i]];
                if (amount > 0) {
                    delete lenders[_loanId][lenderList[i]];

IERC20Upgradeable(lendingAssets[loan.term.fundType]).safeTransfer(lenderList[i], amount); //Repayment of loan assets to the borrower
                }
            }
        } else if (loan.loanType == LoanType.AUCTION) { //Check whether it is an audit debit and credit type

IERC20Upgradeable(lendingAssets[loan.term.fundType]).safeTransfer(loanTopBidder[_loanId], loan.fundRaised);//Repayment of loan assets
```

```
        }

        delete loans[_loanId];
        delete loanLenders[_loanId];
        delete loanTopBidder[_loanId];

        ILoanRegistry(loanRegistry).releaseAsset(_loanId,  address(0));  //Release  of  mortgaged
assets

        emit LoanCancelled(_loanId, loan.borrower);
    }
```

Safety advice：NONE.

8.3.23. Withdrawfund business logic design【security】

Audit description： Conduct security audit on the logic design of withdraw loan business of withdrawfund in the contract to check whether there are security problems such as improper authority control and design defects.

Audit results： The logic design of withdraw loan business of withdrawfund in the contract is correct.

Code file： PeerLoan.sol 139~151

Code information：

```
/// @notice Withdraw borrowed fund from the loan `_loanId`
    ///    only the borrower of the loan can call this
    function withdrawFund(bytes32 _loanId) nonReentrant external { //Prevent reentry attacks
        LoanMetadata storage loan = loans[_loanId]; //According to_ Loanid: get loan information
        if (loan.borrower != _msgSender() || loan.status != LoanStatus.FUNDED) { //Check whether
the caller is the borrower and whether the loan raising status is completed (decide whether it can be
withdrawn)
            revert NotAllowed();
        }
```

```
        loan.status = LoanStatus.WITHDRAWN; //Change loan status
        loan.withdrawAt = block.timestamp; //Change extraction time

        IERC20Upgradeable(lendingAssets[loan.term.fundType]).safeTransfer(loan.borrower,
loan.fundRaised); //Withdrawal of loan

        emit WithdrawFund(_loanId, _msgSender(), loan.fundRaised);
        }
```

Safety advice：NONE.

8.3.24. Logic design of relay service【security】

Audit description： Conduct security audit on the logic design of the repay loan repayment business in the contract to check whether there are security problems such as improper authority control and design defects.

Audit results： The logical design of the repay loan repayment business in the contract is correct.

Code file： PeerLoan.sol 154~196

Code information：

```
/// @notice Repays the loan `_loanId`, only the borrower of the loan can call this
    function repay(bytes32 _loanId) external payable {
        LoanMetadata memory loan = loans[_loanId];//According to_ Loanid: get loan information
        if (loan.borrower != _msgSender() || loan.status != LoanStatus.WITHDRAWN) { //Check
whether the caller is a borrower and whether the loan status is withdrawn
            revert NotAllowed();
        }

        address assetAddr = lendingAssets[loan.term.fundType]; //Retrieve loan type
        if (assetAddr == address(0)) { //Check whether assetaddr is an empty address
            revert InvalidFundType();
        }
```

```
        uint256 interest = interestOwed(_loanId);    //Calculate interest
        uint256 repayAmount = loan.withdrawAt == 0 ? loan.fundRaised : loan.fundRaised + interest;
//Calculate the amount of assets to be repaid
        if  (IERC20Upgradeable(assetAddr).balanceOf(_msgSender())  <  repayAmount)  {  //Check
whether the repayment amount is higher than the assets currently owned by Brower
            revert InsufficientBalance();
        }

        IERC20Upgradeable(assetAddr).safeTransferFrom(_msgSender(),                address(this),
repayAmount); //Repayment of assets

        if (loan.loanType == LoanType.P2M) { //Check whether the loan type is p2m
            address[] memory lenderList = loanLenders[_loanId]; //Retrieve lender list
            uint256 size = lenderList.length;
            for (uint256 i = 0; i < size; i++) {
                uint256 principle = lenders[_loanId][lenderList[i]];
                if (principle > 0) {
                    delete lenders[_loanId][lenderList[i]];
                    uint256  amount  =  principle  +  (principle  *  SCALAR  /  loan.fundRaised  *
(interest - computeProtocolFee(interest))) / SCALAR;

IERC20Upgradeable(lendingAssets[loan.term.fundType]).safeTransfer(lenderList[i],            amount);
//repayment
                }
            }
        } else if (loan.loanType == LoanType.AUCTION) { //Check whether the loan type is "audit"
            uint256 amount = loan.fundRaised + (interest - computeProtocolFee(loan.fundRaised));
//Clearing repayment after deducting the agreement fee

IERC20Upgradeable(lendingAssets[loan.term.fundType]).safeTransfer(loanTopBidder[_loanId],
amount);
        }

        delete loans[_loanId];
        delete loanLenders[_loanId];
        delete loanTopBidder[_loanId];
```

```
        ILoanRegistry(loanRegistry).releaseAsset(_loanId,  address(0));  //Release  of  mortgaged
assets

        emit LoanRepaid(_loanId, loan.borrower, repayAmount);
        }
```

Safety advice：NONE.

8.3.25. Business logic design of addcollaborators【security】

Audit description： Conduct security audit on the business logic design of addcollaborators in the contract to check whether there are security problems such as improper permission control and design defects.

Audit results： The business logic design of addcollaborators in the contract is correct.

Code file： TokenVault.sol 57~69

Code information：

```
    /// @notice Records the secured NFTs to the corresponding loan.
    function  addCollaterals(bytes32  _loanId,  DataTypes.CollateralizedAsset[]  calldata  _assets)
external onlyLoanManager { //Only manager calls are allowed
        loans[_loanId] = securedAssetKey; //Set the secured asset key

        uint256 i = 0;
        uint256 size = _assets.length;
        for (; i < size; i++) {
            DataTypes.CollateralizedAsset memory ca = DataTypes.CollateralizedAsset({
                assetAddress: _assets[i].assetAddress,
                assetIds: _assets[i].assetIds
            });
            securedAssetsInTheLoan[securedAssetKey++].push(ca);
        }
```

```
}
```

Safety advice：NONE.

8.3.26. Logic design of airdrop distribution【security】

Audit description： Conduct security audit on the business logic design of investment and distribution in the contract to check whether there are security problems such as improper authority control and design defects.

Audit results： The business logic design of investment and distribution in the contract is correct.

Code file： TokenVault.sol 92~98

Code information：

```
    /// @notice Claims all kind of ERC721 tokens that airdropped to the tokens belongs to this vault.
    /// @param _addr The ERC721 token address
    /// @param _tknId The token ID to be claimed
    function claimERC721Airdrop(address _addr, uint256 _tknId)
        external
        onlyOwner
        onlyIfTheAssetIsNotInTheLoan(_addr, _tknId)
    { //Only the owner of the contract is allowed to call
        IERC721(_addr).safeTransferFrom(address(this), owner, _tknId); //Airdrop
    }
    modifier onlyIfTheAssetIsNotInTheLoan(address _addr, uint256 _tknId) { //When checking whether the asset is in the loan
        bool inSecured = _isAssetInSecured(_addr, _tknId); //Check whether there is guarantee
        if (inSecured) {
            revert AssetIsInCollateral({ addr: _addr, tknId: _tknId });
        }
        _;
    }
    /// @notice Claims all kind of ERC1155 tokens that airdropped to the tokens belongs to this vault.
    /// @param _addr The ERC1155 token address
```

```
    /// @param _tknId The token ID to be claimed
    /// @param _amount The amount of token to be claimed
    function claimERC1155Airdrop(address _addr, uint256 _tknId, uint256 _amount)
        external
        onlyOwner
        onlyIfTheAssetIsNotInTheLoan(_addr, _tknId)
    {//Only the owner of the contract is allowed to call
        IERC1155(_addr).safeTransferFrom(address(this), owner, _tknId, _amount, ""); //Release airdrop
    }
    /// @notice Claims all kind of ERC20 tokens that airdropped to the tokens belongs to this vault.
    /// @param _addr The ERC20 token address to be claimed
    function claimERC20Airdrop(address _addr) external onlyOwner { //Only the owner of the contract is allowed to call
        bool succeeded = ERC20(_addr).transfer(owner, ERC20(_addr).balanceOf(address(this)));
        if (!succeeded) {
            revert ErrorToClaimTheToken({ symbol: ERC20(_addr).symbol() });
        }
    }
```

Safety advice：NONE.

8.3.27. Contract authority concentration detection【security】

Audit description： Detect the concentration of authority in the contract and check whether the relevant business logic is reasonable.

Audit results： Contract authority concentration detection has been repaired.

Code file： NamaToken.sol 30~40

BaseLoad.sol 82~104

FundingPool.sol 158~163

RewardsDistributionRecipient.sol 17~20

Code information：

NamaToken.sol 30~40

```
    function snapshot() external onlyOwner {
        _snapshot(); //Create Snapshot
    }

    function setPause(bool _flag) external onlyOwner {
        if (_flag) {
            _pause(); //suspend
        } else {
            _unpause(); //Enable
        }
    }
```

RewardsDistributionRecipient.sol 17~20

```
abstract contract RewardsDistributionRecipient is Owned {
    address public rewardsDistribution; //Award allocation address

    function notifyRewardAmount(uint256 reward) virtual external;

    modifier onlyRewardsDistribution() {
        require(_msgSender() == rewardsDistribution, "Caller is not RewardsDistribution contract");
        _;
    }

    function setRewardsDistribution(address _rewardsDistribution) external onlyOwner
{          rewardsDistribution = _rewardsDistribution;
    }
}
```

BaseLoad.sol 82~104

```
    function setLoanRegistry(address _creatorAddr) external onlyOwner {
        loanRegistry = _creatorAddr;
    }

    function setProtocolFeeRate(uint256 _rate) external onlyOwner {
        protocolFeeRate = _rate;
    }
```

```
    function setLiquidationAddress(address _addr) external onlyOwner {
        liquidationAddress = _addr;
    }

    function setGracePeriod(uint256 _gp) external onlyOwner {
        gracePeriod = _gp;
    }

    function pause() public onlyOwner {
        _pause();
    }

    function unpause() public onlyOwner {
        _unpause();
    }
```

FundingPool.sol 158~163

```
    /* ========== RESTRICTED FUNCTIONS ========== */

    function setController(address _controller, bool _status) external onlyOwner {
        require(_controller != address(0), "zero address");
        controllers[_controller] = _status;
    }
```

LoanRegistry.sol 131~147

```
    /// @dev Updates the peer loan contract address
    function setPeerLoan(address _peerAddr) external onlyOwner {
        peerLoan = _peerAddr;
    }

    /// @dev Updates the pool loan contract address
    function setPoolLoan(address _poolAddr) external onlyOwner {
        poolLoan = _poolAddr;
    }

    function pause() public onlyOwner {
```

```
        _pause();
    }

    function unpause() public onlyOwner {
        _unpause();
    }
```

Safety advice：NONE.

9. Contract source code

BaseLoan.sol

```
// SPDX-License-Identifier: BUSL-1.1

pragma solidity 0.8.9;

import "hardhat/console.sol";

import {Initializable} from "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
import {OwnableUpgradeable} from "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";
import {PausableUpgradeable} from "@openzeppelin/contracts-upgradeable/security/PausableUpgradeable.sol";
import {ReentrancyGuardUpgradeable} from "@openzeppelin/contracts-upgradeable/security/ReentrancyGuardUpgradeable.sol";
import {UUPSUpgradeable} from "@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol";

import {IBaseLoan} from "./interfaces/IBaseLoan.sol";
import {ILiquidation} from "./interfaces/ILiquidation.sol";


interface ILoanRegistry {
    function releaseAsset(bytes32 _loanId, address _to) external;
}
```

```
/// @author Victor F <victor@nama.finance>
abstract contract BaseLoan is
    Initializable,
    IBaseLoan,
    OwnableUpgradeable,
    PausableUpgradeable,
    ReentrancyGuardUpgradeable,
    UUPSUpgradeable
{

    uint256 public constant SCALAR = 10000;
    uint256 public gracePeriod;
    uint256 public protocolFeeRate;
    address public liquidationAddress;
    address public loanRegistry;

    mapping(bytes32 => LoanMetadata) public loans;
    mapping(bytes32 => address) public lendingAssets;
    mapping(bytes32 => mapping(address => uint256)) public lenders;

    modifier onlyCreator() {
        if (loanRegistry != _msgSender()) {
            revert NotAllowed();
        }
        _;
    }

    modifier onlyManager() {
        if (loanRegistry != _msgSender() || owner() != _msgSender()) {
            revert NotAllowed();
        }
        _;
    }

    /// @notice Liquidate the loan, anybody can call this and get reward if it succeeded
    function liquidateLoan(bytes32 _loanId) external {
```

```
        LoanMetadata storage loan = loans[_loanId];
        if (liquidationAddress == address(0)) {
            revert ZeroAddress();
        }
        if (block.timestamp <= (loan.withdrawAt + loan.term.duration + gracePeriod)) {
            revert NotAllowed();
        }

        loan.status = LoanStatus.DEFAULTED;

        ILoanRegistry(loanRegistry).releaseAsset(_loanId, liquidationAddress);

        ILiquidation(liquidationAddress).liquidate(_loanId);

        emit Liquidated(_loanId, _msgSender());
    }

    function withdrawNativeToken(uint256 _amount) public onlyOwner {
        payable(_msgSender()).transfer(_amount == 0 ? address(this).balance : _amount);
    }

    function addLendingAsset(bytes32 _fundType, address _assetAddr) external onlyOwner {
        lendingAssets[_fundType] = _assetAddr;
    }

    function setLoanRegistry(address _creatorAddr) external onlyOwner {
        loanRegistry = _creatorAddr;
    }

    function setProtocolFeeRate(uint256 _rate) external onlyOwner {
        protocolFeeRate = _rate;
    }

    function setLiquidationAddress(address _addr) external onlyOwner {
        liquidationAddress = _addr;
    }
```

```
function setGracePeriod(uint256 _gp) external onlyOwner {
        gracePeriod = _gp;
}

function pause() public onlyOwner {
        _pause();
}

function unpause() public onlyOwner {
        _unpause();
}

function getLoanData(bytes32 _loanId) external view returns (LoanMetadata memory) {
        return loans[_loanId];
}

function getLoanStatus(bytes32 _loanId) external view returns (LoanStatus) {
        return loans[_loanId].status;
}

function loanExpirationTime(bytes32 _loanId) external view returns (uint256) {
        LoanMetadata memory loan = loans[_loanId];

        return loan.term.duration + loan.withdrawAt;
}

function totalOwed(bytes32 _loanId) external view returns (uint256) {
        LoanMetadata memory loan = loans[_loanId];
        if (loan.withdrawAt == 0 || loan.status != LoanStatus.WITHDRAWN) return 0;

        return loan.fundRaised + interestOwed(_loanId);
}

function interestOwed(bytes32 _loanId) public view returns (uint256) {
        LoanMetadata memory loan = loans[_loanId];
```

```
        if (loan.withdrawAt == 0 || loan.status != LoanStatus.WITHDRAWN) return 0;

        return _interestOwed(
            loan.fundRaised - loan.amountRepaid,
            loan.term.interestRate,
            block.timestamp - loan.withdrawAt,
            loan.term.duration
        );
    }

    function _interestOwed(
        uint256 _borrowAmount,
        uint256 _loanInterestRate,
        uint256 _loanDurationToNow,
        uint256 _loanExpirationTime
    )
        internal
        pure
        returns (uint256)
    {
        if (_loanDurationToNow > _loanExpirationTime) {
            _loanDurationToNow = _loanExpirationTime;
        }

        uint256 toDateInDays = _loanDurationToNow / 1 days;
        if (toDateInDays == 0) {
            toDateInDays = 1;
        } else if (_loanDurationToNow % 1 days != 0) {
            toDateInDays += 1;
        }

        uint256 interestPerDay = (_borrowAmount * _loanInterestRate) / SCALAR / 100 / 365;
        return interestPerDay * toDateInDays;
    }

    function computeProtocolFee(uint256 interest) internal view returns (uint256) {
```

```
        return (interest * protocolFeeRate) / SCALAR / 100;
    }

    function getChainID() internal view returns (uint256) {
        uint256 id;
        assembly {
            id := chainid()
        }
        return id;
    }

    function _authorizeUpgrade(address newImplementation) internal onlyOwner override {}
}
```

FundingPool.sol

```
// SPDX-License-Identifier: BUSL-1.1

pragma solidity 0.8.9;

import "hardhat/console.sol";

import {ReentrancyGuard} from "@openzeppelin/contracts/security/ReentrancyGuard.sol";
import {Pausable} from "@openzeppelin/contracts/security/Pausable.sol";

import {ERC4626} from "@rari-capital/solmate/src/mixins/ERC4626.sol";
import {ERC20} from "@rari-capital/solmate/src/tokens/ERC20.sol";
import {SafeTransferLib} from "@rari-capital/solmate/src/utils/SafeTransferLib.sol";

import {IStakingRewards} from "./interfaces/IStakingRewards.sol";
import {RewardsDistributionRecipient, Owned} from "./interfaces/RewardsDistributionRecipient.sol";


/// @title The funding pool
/// @dev A folk of Synthetix's StakingRewards with the implementation of tokenized Vault of ERC4626
contract FundingPool is IStakingRewards, ERC4626, RewardsDistributionRecipient, Pausable,
```

```
ReentrancyGuard {
    error NotAllowedOperator();

    using SafeTransferLib for ERC20;

    /* ========== STATE VARIABLES ========== */

    uint256 public periodFinish = 0;
    uint256 public rewardRate = 1 * 10 ** 18; // 1 reward token per sec
    uint256 public rewardsDuration = 365 days;
    uint256 public lastUpdateTime;
    uint256 public rewardPerTokenStored;

    mapping(address => uint256) public userRewardPerTokenPaid;
    mapping(address => uint256) public rewards;
    mapping(address => bool) public controllers;

    /* ========== EVENTS ========== */

    event RewardAdded(uint256 reward);
    event RewardPaid(address indexed user, uint256 reward);
    event Recovered(address indexed token, uint256 amount);
    event RewardsDurationUpdated(uint256 newDuration);
    event PaidOut(address indexed recipient, uint256 amount);

    /* ========== IMMUTABLES ========== */

    ERC20 public immutable rewardsToken;

    /* ========== MODIFIERS ========== */

    modifier updateReward(address account) {
        rewardPerTokenStored = rewardPerToken();
        lastUpdateTime = block.timestamp;
        if (account != address(0)) {
            rewards[account] = earned(account);
```

```
            userRewardPerTokenPaid[account] = rewardPerTokenStored;
        }
        _;
    }

    modifier onlyControllers() {
        if (!controllers[_msgSender()]) {
            revert NotAllowedOperator();
        }
        _;
    }

    /* ========== CONSTRUCTOR ========== */

    constructor(
        ERC20 _rewardsToken,
        ERC20 _stakingToken,
        address _rewardsDistribution,
        string memory _name,
        string memory _symbol
    ) ERC4626(_stakingToken, _name, _symbol) Owned(_msgSender()) {
        rewardsToken = _rewardsToken;
        rewardsDistribution = _rewardsDistribution;
    }

    /* ========== VIEWS ========== */

    /// @dev Returns the totalSupply if the rewards token is same as the staking token ,
    ///    otherwise returns the balance of this contract in the staking token contract
    function totalAssets() public view override returns (uint256) {
        return rewardsToken == asset ? totalSupply : ERC20(asset).balanceOf(address(this));
    }

    /// @dev Returns the end of time of the reward
    function lastTimeRewardApplicable() public view returns (uint256) {
        return block.timestamp < periodFinish ? block.timestamp : periodFinish;
```

```
    }

    function rewardPerToken() public view returns (uint256) {
        if (totalSupply == 0) {
            return rewardPerTokenStored;
        }
        return rewardPerTokenStored + (((block.timestamp - lastUpdateTime) * rewardRate * 1e18) /
totalSupply);
    }

    /// @dev Returns the total earned of an account
    function earned(address _account) public view returns (uint256) {
        return ((balanceOf[_account] * (rewardPerToken() - userRewardPerTokenPaid[_account])) /
1e18) + rewards[_account];
    }

    /* ========== MUTATIVE FUNCTIONS ========== */

    function deposit(uint256 assets, address receiver)
        public
        override
        nonReentrant
        whenNotPaused
        updateReward(_msgSender())
        returns (uint256 shares)
    {
        shares = super.deposit(assets, receiver);
    }

    function mint(uint256 shares, address receiver)
        public
        override
        nonReentrant
        whenNotPaused
        updateReward(_msgSender())
        returns (uint256 assets)
```

```
    {
        assets = super.mint(shares, receiver);
    }

    function withdraw(
        uint256 _assets,
        address _receiver,
        address _owner
    ) public override nonReentrant whenNotPaused updateReward(_msgSender()) returns (uint256 shares) {
        shares = super.withdraw(_assets, _receiver, _owner);
    }

    function redeem(
        uint256 _shares,
        address _receiver,
        address _owner
    ) public override nonReentrant whenNotPaused updateReward(_msgSender()) returns (uint256 assets) {
        assets = super.redeem(_shares, _receiver, _owner);
    }

    function exit() external {
        withdraw(balanceOf[_msgSender()], _msgSender(), _msgSender());
        getReward();
    }

    // TODO need to check the status of the loan before payout
    function payout(bytes32 /**_loanId**/, address _to, uint256 _amount) external onlyControllers returns (bool) {
        ERC20(asset).safeTransfer(_to, _amount);
        emit PaidOut(_to, _amount);
        return true;
    }

    /* ========== RESTRICTED FUNCTIONS ========== */
```

```
    function setController(address _controller, bool _status) external onlyOwner {
        require(_controller != address(0), "zero address");
        controllers[_controller] = _status;
    }

    function getReward() public nonReentrant updateReward(_msgSender()) {
        uint256 reward = rewards[_msgSender()];
        if (reward > 0) {
            rewards[_msgSender()] = 0;
            rewardsToken.safeTransfer(_msgSender(), reward);
            emit RewardPaid(_msgSender(), reward);
        }
    }

    function pause() public whenNotPaused onlyOwner {
        _pause();
    }

    function unpause() public whenPaused onlyOwner {
        _unpause();
    }

    function setRewardRate(uint256 _rewardRate) external onlyRewardsDistribution {
        rewardRate = _rewardRate;
    }

    function notifyRewardAmount(uint256 reward) external override onlyRewardsDistribution
updateReward(address(0)) {
        if (block.timestamp >= periodFinish) {
            rewardRate = reward / rewardsDuration;
        } else {
            uint256 remaining = periodFinish - block.timestamp;
            uint256 leftover = remaining * rewardRate;
            rewardRate = (reward + leftover) / rewardsDuration;
        }
```

```
        // Ensure the provided reward amount is not more than the balance in the contract.
        // This keeps the reward rate in the right range, preventing overflows due to
        // very high values of rewardRate in the earned and rewardsPerToken functions;
        // Reward + leftover must be less than 2^256 / 10^18 to avoid overflow.
        uint balance = rewardsToken.balanceOf(address(this));
        require(rewardRate <= balance / rewardsDuration, "Provided reward too high");

        lastUpdateTime = block.timestamp;
        periodFinish = block.timestamp + rewardsDuration;
        emit RewardAdded(reward);
    }

    function setRewardsDuration(uint256 _rewardsDuration) external onlyOwner {
        require(
            block.timestamp > periodFinish,
            "Previous rewards period must be complete before changing the duration for the new
period"
        );
        rewardsDuration = _rewardsDuration;
        emit RewardsDurationUpdated(rewardsDuration);
    }

    // Added to support recovering LP Rewards from other systems such as BAL to be distributed to
holders
    function recoverERC20(address tokenAddress, uint256 tokenAmount) external onlyOwner {
        require(tokenAddress != address(asset), "Cannot withdraw the staking token");
        ERC20(tokenAddress).safeTransfer(owner, tokenAmount);
        emit Recovered(tokenAddress, tokenAmount);
    }
}
```

LoanRegistry.sol

```
// SPDX-License-Identifier: BUSL-1.1

pragma solidity 0.8.9;
```

```
import "hardhat/console.sol";

import {OwnableUpgradeable} from "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";
import {Initializable} from "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
import {PausableUpgradeable} from "@openzeppelin/contracts-upgradeable/security/PausableUpgradeable.sol";
import {UUPSUpgradeable} from "@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol";
import {ReentrancyGuardUpgradeable} from "@openzeppelin/contracts-upgradeable/security/ReentrancyGuardUpgradeable.sol";

import {TokenVault, DataTypes, IERC721} from "./TokenVault.sol";
import {IBaseLoan} from "./interfaces/IBaseLoan.sol";


/// @author Victor F <victor@nama.finance>
contract LoanRegistry is
    Initializable,
    OwnableUpgradeable,
    PausableUpgradeable,
    ReentrancyGuardUpgradeable,
    UUPSUpgradeable,
    DataTypes
{
    address public peerLoan;
    address public poolLoan;

    mapping(address => bool) public allowedAssets; // allowed NFT assets for the pool based loans
    mapping(address => address) public tokenVaults; // borrower => token vault
    mapping(bytes32 => address) public loanVaults; // loanId => token vault

    event LoanEvent(address indexed loanAddr, address indexed borrower, uint256 indexed status);

    /// @custom:oz-upgrades-unsafe-allow constructor
    constructor() initializer {}
```

```
function initialize(address _peerLoan, address _poolLoan) public initializer {
        __Ownable_init();
        __Pausable_init();
        peerLoan = _peerLoan;
        poolLoan = _poolLoan;
}

/// @dev Only the PeerLoan and PoolLoan can call the function
modifier onlyNamaLoans() {
        if (_msgSender() != peerLoan && _msgSender() != poolLoan) {
                revert NotAllowed();
        }
        _;
}

/// @notice Creates a loan
function createLoan(
        bytes32 _loanId,
        LoanTerm memory _term,
        LoanType _loanType,
        CollateralizedAsset[] memory _assets,
        string calldata _destRecvChain,
        string calldata _recverAddress
)
        external
        payable
        whenNotPaused
        nonReentrant
{
        if (_loanId.length == 0 || _term.duration == 0 || _assets.length == 0
                || _term.loanAmount == 0 || _term.targetAmount < _term.loanAmount) {
                revert ArgumentException();
        }

        if (_loanType != LoanType.P2M && _loanType != LoanType.AUCTION && _loanType !=
```

```
LoanType.POOL) {
                revert NotSupportedLoanException();
            }

            address vaultAddr = tokenVaults[_msgSender()];
            if (_loanType == LoanType.POOL) {
                // validate whether the assets are in the allowlist
                bool valid = _validateAssets(_assets);
                if (!valid) {
                    revert NoSupportedNFTsException();
                }

                TokenVault vault = vaultAddr == address(0) ? new TokenVault(_msgSender()) :
TokenVault(vaultAddr);
                bool succeed = _depositNFTs(_assets, _msgSender(), address(vault));
                if (!succeed) {
                    revert ErrorToDepositNFTs();
                }
                vault.addCollaterals(_loanId, _assets);
                IBaseLoan(poolLoan).createLoan{value: msg.value}(
                    _loanId,
                    _msgSender(),
                    _term,
                    _loanType,
                    _destRecvChain,
                    _recverAddress
                );
                loanVaults[_loanId] = address(vault);
            } else {
                TokenVault vault = vaultAddr == address(0) ? new TokenVault(_msgSender()) :
TokenVault(vaultAddr);
                bool succeed = _depositNFTs(_assets, _msgSender(), address(vault));
                if (!succeed) {
                    revert ErrorToDepositNFTs();
                }
                vault.addCollaterals(_loanId, _assets);
```

```
            IBaseLoan(peerLoan).createLoan(
                _loanId,
                _msgSender(),
                _term,
                _loanType,
                _destRecvChain,
                _recverAddress
            );
            loanVaults[_loanId] = address(vault);
        }
    }

    /// @notice Releases the assets locked in the loan `_loanId`
    function releaseAsset(bytes32 _loanId, address _to) external onlyNamaLoans {
        bool succeed = TokenVault(loanVaults[_loanId]).releaseAsset(_loanId, _to);
        if (!succeed) {
            revert ErrorToReleaseAsset();
        }
    }

    /// @notice Adds the asset `_asset` into the allowlist to create pool based loans
    function addAllowedAsset(address _asset, bool _val) external onlyOwner {
        allowedAssets[_asset] = _val;
    }

    /// @dev Updates the peer loan contract address
    function setPeerLoan(address _peerAddr) external onlyOwner {
        peerLoan = _peerAddr;
    }

    /// @dev Updates the pool loan contract address
    function setPoolLoan(address _poolAddr) external onlyOwner {
        poolLoan = _poolAddr;
    }

    function pause() public onlyOwner {
```

```
        _pause();
    }

    function unpause() public onlyOwner {
        _unpause();
    }

    /// @notice Validates whether the deposited assets are in the allowed list
    function _validateAssets(CollateralizedAsset[] memory _assets) internal view returns (bool) {
        uint256 size = _assets.length;
        for (uint i = 0; i < size; i++) {
            if (allowedAssets[_assets[i].assetAddress]) {
                return true;
            }
        }
        return false;
    }

    /// @notice Deposits NFTs to the corresponding token vault
    function _depositNFTs(CollateralizedAsset[] memory _assets, address _sender, address _recipient)
internal returns (bool) {
        // make sure all NFTs must be approved before transferring
        uint256 size = _assets.length;
        uint256 i = 0;
        for (; i < size; i++) {
            address _nftAddr = _assets[i].assetAddress;
            uint256[] memory assetIds = _assets[i].assetIds;
            uint256 tokenSize = assetIds.length;

            for (uint256 j = 0; j < tokenSize; j++) {
                IERC721(_nftAddr).safeTransferFrom(_sender, _recipient, assetIds[j]);
            }
        }

        return true;
    }
```

```
    function _authorizeUpgrade(address newImplementation) internal override onlyOwner {}
}
```

PeerLoan.sol

```
// SPDX-License-Identifier: BUSL-1.1

pragma solidity 0.8.9;

import "hardhat/console.sol";

import {IERC20Upgradeable} from "@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol";
import {SafeERC20Upgradeable} from "@openzeppelin/contracts-upgradeable/token/ERC20/utils/SafeERC20Upgradeable.sol";
import {StringsUpgradeable} from "@openzeppelin/contracts-upgradeable/utils/StringsUpgradeable.sol";

import {BaseLoan, ILoanRegistry} from "./BaseLoan.sol";

/// @author Victor F <victor@nama.finance>
contract PeerLoan is BaseLoan {
    using SafeERC20Upgradeable for IERC20Upgradeable;

    mapping(bytes32 => address) public loanTopBidder;
    mapping(bytes32 => address[]) public loanLenders;

    /// @custom:oz-upgrades-unsafe-allow constructor
    constructor() initializer {}

    function initialize() public initializer {
        __Ownable_init();
        __Pausable_init();
        gracePeriod = 24 hours;
    }
```

```
    function createLoan(
        bytes32 _loanId,
        address _borrower,
        LoanTerm memory _loanTerm,
        LoanType _loanType,
        string calldata _destRecvChain,
        string calldata _recverAddress
    )
        external
        payable
        onlyCreator
    {
        if (lendingAssets[_loanTerm.fundType] == address(0)) {
            revert InvalidFundType();
        }

        LoanMetadata storage loan = loans[_loanId];
        if (loan.appliedAt != 0) {
            revert LoanAlreadyExisted();
        }

        loan.loanId = _loanId;
        loan.term = _loanTerm;
        loan.term.duration = _loanTerm.duration * 1 days;
        loan.appliedAt = block.timestamp;
        loan.loanType = _loanType;
        loan.status = LoanStatus.AWAITING;
        loan.borrower = _borrower;
        loan.createdOnChain = StringsUpgradeable.toString(block.chainid);

        loan.destRecvChain = _destRecvChain;
        loan.recverAddress = _recverAddress;

        emit LoanCreated(loan.loanId, loan.borrower, lendingAssets[loan.term.fundType],
loan.term.loanAmount);
    }
```

```
    /// @notice lend an amount of `_amount` to the loan `_loanId`,
    ///    only the lender of the loan can call this
    function fund(bytes32 _loanId, uint256 _amount) external payable nonReentrant whenNotPaused
{
        LoanMetadata memory loan = loans[_loanId];
        if (loan.borrower == _msgSender() || loan.status != LoanStatus.AWAITING || _amount == 0)
{
            revert NotAllowed();
        }

        if    (IERC20Upgradeable(lendingAssets[loan.term.fundType]).balanceOf(_msgSender())    <
_amount) {
            revert InsufficientBalance();
        }

        if (loan.loanType == LoanType.P2M) {
            _p2mOffer(_loanId, _amount, loan.term.fundType);
        } else {
            _auctionOffer(_loanId, _amount, loan.term.fundType);
        }
    }

    /// @notice Revokes the offer to a loan `_loanId`,
    ///    only works P2M type of loan and only the lender of the loan can call this
    function revokeOffer(bytes32 _loanId, uint256 _amount) external nonReentrant {
        LoanMetadata storage loan = loans[_loanId];
        if (loan.loanType != LoanType.P2M || _amount == 0
            || _amount > lenders[_loanId][_msgSender()]
            || (loan.status != LoanStatus.AWAITING && loan.status != LoanStatus.FUNDED)) {
            revert NotAllowed();
        }

        loan.fundRaised -= _amount;
        lenders[_loanId][_msgSender()] -= _amount;
```

```
            if (loan.fundRaised < loan.term.targetAmount) {
                loan.status = LoanStatus.AWAITING;
            }
            IERC20Upgradeable(lendingAssets[loan.term.fundType]).safeTransfer(_msgSender(),
_amount);
            emit RevokeOffer(_loanId, _msgSender(), loan.status, _amount, loan.fundRaised);
        }

        /// @notice Cancels the loan application,
        ///    only the borrower of the loan can call this
        function cancelLoanApplication(bytes32 _loanId) external nonReentrant {
            LoanMetadata storage loan = loans[_loanId];
            if (loan.borrower != _msgSender() || loan.status != LoanStatus.AWAITING) {
                revert NotAllowed();
            }

            loan.status = LoanStatus.CANCELLED;

            if (loan.loanType == LoanType.P2M) {
                address[] memory lenderList = loanLenders[_loanId];
                uint256 size = lenderList.length;
                for (uint256 i = 0; i < size; i++) {
                    uint256 amount = lenders[_loanId][lenderList[i]];
                    if (amount > 0) {
                        delete lenders[_loanId][lenderList[i]];

IERC20Upgradeable(lendingAssets[loan.term.fundType]).safeTransfer(lenderList[i], amount);
                    }
                }
            } else if (loan.loanType == LoanType.AUCTION) {

IERC20Upgradeable(lendingAssets[loan.term.fundType]).safeTransfer(loanTopBidder[_loanId],
loan.fundRaised);
            }

            delete loans[_loanId];
```

```
            delete loanLenders[_loanId];
            delete loanTopBidder[_loanId];

            ILoanRegistry(loanRegistry).releaseAsset(_loanId, address(0));

            emit LoanCancelled(_loanId, loan.borrower);
        }

        /// @notice Withdraw borrowed fund from the loan `_loanId`
        ///     only the borrower of the loan can call this
        function withdrawFund(bytes32 _loanId) nonReentrant external {
            LoanMetadata storage loan = loans[_loanId];
            if (loan.borrower != _msgSender() || loan.status != LoanStatus.FUNDED) {
                revert NotAllowed();
            }

            loan.status = LoanStatus.WITHDRAWN;
            loan.withdrawAt = block.timestamp;

            IERC20Upgradeable(lendingAssets[loan.term.fundType]).safeTransfer(loan.borrower,
loan.fundRaised);

            emit WithdrawFund(_loanId, _msgSender(), loan.fundRaised);
        }

        /// @notice Repays the loan `_loanId`, only the borrower of the loan can call this
        function repay(bytes32 _loanId) external payable {
            LoanMetadata memory loan = loans[_loanId];
            if (loan.borrower != _msgSender() || loan.status != LoanStatus.WITHDRAWN) {
                revert NotAllowed();
            }

            address assetAddr = lendingAssets[loan.term.fundType];
            if (assetAddr == address(0)) {
                revert InvalidFundType();
            }
```

```
        uint256 interest = interestOwed(_loanId);
        uint256 repayAmount = loan.withdrawAt == 0 ? loan.fundRaised : loan.fundRaised +
interest;
        if (IERC20Upgradeable(assetAddr).balanceOf(_msgSender()) < repayAmount) {
            revert InsufficientBalance();
        }

        IERC20Upgradeable(assetAddr).safeTransferFrom(_msgSender(), address(this),
repayAmount);

        if (loan.loanType == LoanType.P2M) {
            address[] memory lenderList = loanLenders[_loanId];
            uint256 size = lenderList.length;
            for (uint256 i = 0; i < size; i++) {
                uint256 principle = lenders[_loanId][lenderList[i]];
                if (principle > 0) {
                    delete lenders[_loanId][lenderList[i]];
                    uint256 amount = principle + (principle * SCALAR / loan.fundRaised *
(interest - computeProtocolFee(interest))) / SCALAR;

IERC20Upgradeable(lendingAssets[loan.term.fundType]).safeTransfer(lenderList[i], amount);
                }
            }
        } else if (loan.loanType == LoanType.AUCTION) {
            uint256 amount = loan.fundRaised + (interest - computeProtocolFee(loan.fundRaised));

IERC20Upgradeable(lendingAssets[loan.term.fundType]).safeTransfer(loanTopBidder[_loanId],
amount);
        }

        delete loans[_loanId];
        delete loanLenders[_loanId];
        delete loanTopBidder[_loanId];

        ILoanRegistry(loanRegistry).releaseAsset(_loanId, address(0));
```

```
        emit LoanRepaid(_loanId, loan.borrower, repayAmount);
    }

    /**
     * @notice Gets the principle of the lender `_lender` in a loan `_loanId`.
     *
     * @param _loanId the loan to be retrived.
     * @param _lender the lender.
     */
    function getPrinciple(bytes32 _loanId, address _lender) external view returns (uint256) {
        return lenders[_loanId][_lender];
    }

    function _p2mOffer(bytes32 _loanId, uint256 _amount, bytes32 _fundType) internal {
        LoanMetadata storage loan = loans[_loanId];
        uint256 leftover = loan.term.loanAmount - loan.fundRaised;
        if (_amount > leftover) {
            _amount = leftover;
        }

        loan.fundRaised += _amount;
        lenders[_loanId][_msgSender()] += _amount;
        loanLenders[_loanId].push(_msgSender());

        if (loan.fundRaised == loan.term.loanAmount) {
            loan.status = LoanStatus.FUNDED;
        }

        IERC20Upgradeable(lendingAssets[_fundType]).safeTransferFrom(_msgSender(),
address(this), _amount);

        emit MakeOffer(_loanId, _msgSender(), loan.status, lenders[_loanId][_msgSender()],
loan.fundRaised);
    }
```

```
    function _auctionOffer(bytes32 _loanId, uint256 _amount, bytes32 _fundType) internal {
        LoanMetadata storage loan = loans[_loanId];
        if ((loan.fundRaised == 0 && _amount < loan.term.loanAmount) || _amount <= loan.fundRaised) {
            revert InvalidLoanAmount();
        }

        // if offerred amount is greater than the borrowing amount then set the offer amount to target amount
        if (_amount > loan.term.targetAmount) {
            _amount = loan.term.targetAmount;
        }

        if (loanTopBidder[_loanId] != address(0)) {
            address bidder = loanTopBidder[_loanId];
            uint256 bidAmount = lenders[_loanId][bidder];

            loanTopBidder[_loanId] = address(0);
            lenders[_loanId][bidder] = 0;
            loan.fundRaised -= bidAmount;

            IERC20Upgradeable(lendingAssets[_fundType]).safeTransfer(bidder, bidAmount);
        }

        loan.fundRaised = _amount;
        lenders[_loanId][_msgSender()] = _amount;
        loanTopBidder[_loanId] = _msgSender();

        if (loan.term.targetAmount != 0 && loan.fundRaised == loan.term.targetAmount) {
            loan.status = LoanStatus.FUNDED;
        }

        IERC20Upgradeable(lendingAssets[_fundType]).safeTransferFrom(_msgSender(), address(this), _amount);

        emit MakeOffer(_loanId, _msgSender(), loan.status, lenders[_loanId][_msgSender()],
```

```
_amount);
        }
}
```

TokenVault.sol

```
// SPDX-License-Identifier: BUSL-1.1

pragma solidity 0.8.9;

import {ERC1155Holder} from "@openzeppelin/contracts/token/ERC1155/utils/ERC1155Holder.sol";
import {ERC721Holder} from "@openzeppelin/contracts/token/ERC721/utils/ERC721Holder.sol";
import {ERC20} from "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import {IERC721} from "@openzeppelin/contracts/token/ERC721/IERC721.sol";
import {IERC1155} from "@openzeppelin/contracts/token/ERC1155/IERC1155.sol";

import {Owned} from "./utils/Owned.sol";
import {DataTypes} from "./interfaces/DataTypes.sol";


/// @title The token vault for holding the secured NFTs
/// @notice The borrowers can claim the airdropped tokens (both NFTs and ERC20s) within the validity of the loan
///      P.S. borrowers can use this contract as their token vault forever as only themselvies have the right to
///         claim/withdraw tokens belongs to this contract address
/// @author Victor F <victor@nama.finance>
contract TokenVault is Owned, ERC721Holder, ERC1155Holder {
        /// Not allowed to claim if asset `tknId` in `addr` collection is in collateral.
        /// @param addr The asset contract address.
        /// @param tknId The token id.
        error AssetIsInCollateral(address addr, uint256 tknId);
        error Unauthorised();
        error ErrorToClaimTheToken(string symbol);

        address public immutable loanManager;
```

```
    uint256 private securedAssetKey = 1;

    mapping(bytes32 => uint256) private loans;
    mapping(uint256 => DataTypes.CollateralizedAsset[]) private securedAssetsInTheLoan;

    constructor(address _vaultOwner) Owned(_vaultOwner) {
        loanManager = _msgSender();
    }

    modifier onlyLoanManager() {
        if (_msgSender() != loanManager) {
            revert Unauthorised();
        }

        _;
    }

    modifier onlyIfTheAssetIsNotInTheLoan(address _addr, uint256 _tknId) {
        bool inSecured = _isAssetInSecured(_addr, _tknId);
        if (inSecured) {
            revert AssetIsInCollateral({ addr: _addr, tknId: _tknId });
        }

        _;
    }

    /// @notice Records the secured NFTs to the corresponding loan.
    function addCollaterals(bytes32 _loanId, DataTypes.CollateralizedAsset[] calldata _assets)
external onlyLoanManager {
        loans[_loanId] = securedAssetKey;

        uint256 i = 0;
        uint256 size = _assets.length;
        for (; i < size; i++) {
            DataTypes.CollateralizedAsset memory ca = DataTypes.CollateralizedAsset({
```

```
                assetAddress: _assets[i].assetAddress,
                assetIds: _assets[i].assetIds
            });
            securedAssetsInTheLoan[securedAssetKey++].push(ca);
        }
    }

    /// @notice Releases the secured NFTs in the loan to the borrower when it's get repaid/cancelled,
    ///      or to the liquidation contract when it's get defaulted
    /// @notice This can only be called by the NAMA loan contracts and only if the loanId is still
    /// @param _loanId The NFT contract address
    /// @param _to The recipient, if it's zero address then transfer the NFTs to the owner of this vault.
    function releaseAsset(bytes32 _loanId, address _to)
        external
        onlyLoanManager
        returns (bool)
    {
        _transferNFTs(_loanId, _to == address(0) ? owner : _to);

        delete securedAssetsInTheLoan[loans[_loanId]];
        delete loans[_loanId];

        return true;
    }

    /// @notice Claims all kind of ERC721 tokens that airdropped to the tokens belongs to this vault.
    /// @param _addr The ERC721 token address
    /// @param _tknId The token ID to be claimed
    function claimERC721Airdrop(address _addr, uint256 _tknId)
        external
        onlyOwner
        onlyIfTheAssetIsNotInTheLoan(_addr, _tknId)
    {
        IERC721(_addr).safeTransferFrom(address(this), owner, _tknId);
    }
```

```
    /// @notice Claims all kind of ERC1155 tokens that airdropped to the tokens belongs to this vault.
    /// @param _addr The ERC1155 token address
    /// @param _tknId The token ID to be claimed
    /// @param _amount The amount of token to be claimed
    function claimERC1155Airdrop(address _addr, uint256 _tknId, uint256 _amount)
        external
        onlyOwner
        onlyIfTheAssetIsNotInTheLoan(_addr, _tknId)
    {
        IERC1155(_addr).safeTransferFrom(address(this), owner, _tknId, _amount, "");
    }

    /// @notice Claims all kind of ERC20 tokens that airdropped to the tokens belongs to this vault.
    /// @param _addr The ERC20 token address to be claimed
    function claimERC20Airdrop(address _addr) external onlyOwner {
        bool succeeded = ERC20(_addr).transfer(owner, ERC20(_addr).balanceOf(address(this)));
        if (!succeeded) {
            revert ErrorToClaimTheToken({ symbol: ERC20(_addr).symbol() });
        }
    }

    /// @notice Returns the assets that in collateral
    function getAssetsInTheLoan(bytes32 _loanId) external view returns
(DataTypes.CollateralizedAsset[] memory) {
        return securedAssetsInTheLoan[loans[_loanId]];
    }

    /// @notice Checks whether the `_tokenId` of NFT `_addr` is in the secured loans.
    /// @param _addr The asset collection address
    /// @param _tokenId The token id
    function _isAssetInSecured(address _addr, uint256 _tokenId) private view returns (bool) {
        for (uint256 k = 1; k <= securedAssetKey; k++) {
            DataTypes.CollateralizedAsset[] memory _assets = securedAssetsInTheLoan[k];
            uint256 size = _assets.length;
            for (uint256 i = 0; i < size; i++) {
                address _nftAddr = _assets[i].assetAddress;
```

```
                    if (_nftAddr != _addr) {
                        continue;
                    }

                    uint256[] memory assetIds = _assets[i].assetIds;
                    uint256 tokenSize = assetIds.length;
                    for (uint256 j = 0; j < tokenSize; j++) {
                        if (assetIds[j] == _tokenId) {
                            return true;
                        }
                    }
                }
            }

            return false;
        }

        /// @notice Transfers the NFTs to `_to`
        /// @param _loanId The loan id
        /// @param _to The recipient
        function _transferNFTs(bytes32 _loanId, address _to) private returns (bool) {
            DataTypes.CollateralizedAsset[] memory _assets =
securedAssetsInTheLoan[loans[_loanId]];
            uint256 size = _assets.length;
            for (uint256 i = 0; i < size; i++) {
                address _nftAddr = _assets[i].assetAddress;
                uint256[] memory assetIds = _assets[i].assetIds;

                uint256 tokenSize = assetIds.length;
                for (uint256 j = 0; j < tokenSize; j++) {
                    IERC721(_nftAddr).safeTransferFrom(address(this), _to, assetIds[j]);
                }
            }

            return true;
        }
```

```
}
```

Owned.sol

```
// SPDX-License-Identifier: AGPL-3.0-only

pragma solidity 0.8.9;

import {Context} from "@openzeppelin/contracts/utils/Context.sol";

// https://docs.synthetix.io/contracts/source/contracts/owned
contract Owned is Context {
    address public owner;
    address public nominatedOwner;

    constructor(address _owner) {
        require(_owner != address(0), "Owner address cannot be 0");
        owner = _owner;
        emit OwnerChanged(address(0), _owner);
    }

    function nominateNewOwner(address _owner) external onlyOwner {
        nominatedOwner = _owner;
        emit OwnerNominated(_owner);
    }

    function acceptOwnership() external {
        require(_msgSender() == nominatedOwner, "You must be nominated before you can accept
ownership");
        emit OwnerChanged(owner, nominatedOwner);
        owner = nominatedOwner;
        nominatedOwner = address(0);
    }

    modifier onlyOwner {
        _onlyOwner();
        _;
    }
```

```
    function _onlyOwner() private view {
        require(_msgSender() == owner, "Only the contract owner may perform this action");
    }

    event OwnerNominated(address newOwner);
    event OwnerChanged(address oldOwner, address newOwner);
}
```

StringToAddress.sol

```
// SPDX-License-Identifier: MIT

pragma solidity 0.8.9;

library StringToAddress {
    function toAddress(string memory _a) internal pure returns (address) {
        bytes memory tmp = bytes(_a);
        if (tmp.length != 42) return address(0);
        uint160 iaddr = 0;
        uint8 b;
        for (uint256 i = 2; i < 42; i++) {
            b = uint8(tmp[i]);
            if ((b >= 97) && (b <= 102)) b -= 87;
            else if ((b >= 65) && (b <= 70)) b -= 55;
            else if ((b >= 48) && (b <= 57)) b -= 48;
            else return address(0);
            iaddr |= uint160(uint256(b) << ((41 - i) << 2));
        }
        return address(iaddr);
    }
}

library AddressToString {
    function toString(address a) internal pure returns (string memory) {
        bytes memory data = abi.encodePacked(a);
        bytes memory characters = "0123456789abcdef";
        bytes memory byteString = new bytes(2 + data.length * 2);
```

```
            byteString[0] = "0";
            byteString[1] = "x";

            for (uint256 i; i < data.length; ++i) {
                byteString[2 + i * 2] = characters[uint256(uint8(data[i] >> 4))];
                byteString[3 + i * 2] = characters[uint256(uint8(data[i] & 0x0f))];
            }
            return string(byteString);
        }
}
```

NamaToken.sol

```
//SPDX-License-Identifier: BUSL-1.1

pragma solidity 0.8.9;

import {ERC20} from "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import {ERC20Burnable} from "@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol";
import {ERC20Pausable} from "@openzeppelin/contracts/token/ERC20/extensions/ERC20Pausable.sol";
import {ERC20Snapshot} from "@openzeppelin/contracts/token/ERC20/extensions/ERC20Snapshot.sol";
import {Ownable} from "@openzeppelin/contracts/access/Ownable.sol";

/// 1e9 * 1e18 1B max
/// @notice This is the NAMA token contract
contract NamaToken is ERC20, ERC20Pausable, ERC20Burnable, ERC20Snapshot, Ownable {
    error ZeroAddress();
    error NotAllowed();

    mapping(address => bool) public minters;

    event AddMinter(address indexed minter);
    event RemoveMinter(address indexed minter);
```

```
constructor() ERC20("NAMA Token", "NAMA") {
        _mint(_msgSender(), 1 * 10 ** decimals());
}

function burn(address _addr, uint256 _amount) external onlyOwner {
        _burn(_addr, _amount);
}

function snapshot() external onlyOwner {
        _snapshot();
}

function setPause(bool _flag) external onlyOwner {
        if (_flag) {
                _pause();
        } else {
                _unpause();
        }
}

function mint(address _to, uint256 _amount) whenNotPaused external {
        if (_msgSender() != owner() && !minters[_msgSender()]) {
                revert NotAllowed();
        }

        _mint(_to, _amount);
}

function addMinter(address _addr) whenNotPaused external onlyOwner {
        if (_addr == address(0)) {
                revert ZeroAddress();
        }

        minters[_addr] = true;
        emit AddMinter(_addr);
```

```
    }

    function removeMinter(address _addr) whenNotPaused external onlyOwner {
        if (_addr == address(0)) {
            revert ZeroAddress();
        }

        delete minters[_addr];
        emit RemoveMinter(_addr);
    }

    function _beforeTokenTransfer(address from, address to, uint256 amount)
        internal
        whenNotPaused
        override(ERC20, ERC20Pausable, ERC20Snapshot)
    {
        super._beforeTokenTransfer(from, to, amount);
    }
```

10. Appendix:Analysis tools

10.1.Solgraph

Solgraph is used to generate a graph of the call relationship between smart contract functions, which is convenient for quickly understanding the call relationship between smart contract functions.

Project address： https://github.com/raineorshine/solgraph

10.2.Sol2uml

Sol2uml is used to generate the calling relationship between smart contract functions in the form of UML diagram.

Project address： https://github.com/naddison36/sol2uml

10.3.Remix-ide

Remix is a browser based compiler and IDE that allows users to build contracts and debug transactions using the solid language.

Project address： http://remix.ethereum.org

10.4.Ethersplay

Etherplay is a plug-in for binary ninja. It can be used to analyze EVM bytecode and graphically present the function call process.

Project address： https://github.com/crytic/ethersplay

10.5.Mythril

Mythril is a security audit tool for EVM bytecode, and supports online contract

audit.

Project address：https://github.com/ConsenSys/mythril

10.6.Echidna

Echidna is a security audit tool for EVM bytecode. It uses fuzzy testing technology and supports integrated use with truss.

Project address：https://github.com/crytic/echidna

11. DISCLAIMERS

Chainlion only issues this report on the facts that have occurred or existed before the issuance of this report, and assumes corresponding responsibilities. For the facts occurring or existing after the issuance, chainlion cannot judge the security status of its smart contract, and is not responsible for it. The security audit analysis and other contents in this report are only based on the documents and materials provided by the information provider to chainlion as of the issuance of this report. Chainlion assumes that the information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed or reflected inconsistent with the actual situation, chainlion shall not be liable for the losses and adverse effects caused thereby. Chainlion only conducted

the agreed safety audit on the safety of the project and issued this report. Chainlion is not responsible for the background and other conditions of the project.