Security Audit Report

NEST Protocol (V3)

SECBIT

Jul 12, 2020

1. Introduction

NEST Protocol (V3) is an Ethereum-based distributed price oracle network protocol. SECBIT Labs conducted an audit from Apr 28th to Jul 12th, 2020, including an analysis of Smart Contracts in 3 areas: **code bugs**, **logic flaws**, and **risk assessment**. The audit results show that NEST Protocol (V3) has no critical security risks. The SECBIT team has some tips on logical implementation, potential risks, and code revising (see part 4 for details).

Type	Description	Level	Status
Logical Implementation	4.3.1 In Nest_3_NestAbonus : reloadTimeAndMapping() , `_snapshot` cannot provide the function of judging whether the snapshot is completed.	Info	Fixed
Logical Implementation	4.3.2 In Nest_3_VoteFactory : changeStateOfEmergency() , the implementation logic of emergency state modification is in doubt.	Low	Fixed
Logical Implementation	In Nest_3_VoteContract : constructor(), when the voting contract is in emergency state, the calculation of the number of total votes and passed votes does not match the actual one.	Low	Fixed
Logical Implementation	In Nest_3_OfferPrice and Nest_NToken_OfferPrice contracts, the update of `blockAddress` in the contract is inconsistent with the contract design document.	Info	Fixed
Logical Implementation	4.3.5 In Nest_3_NestAbonus contract, when the expected bonus increment ratio changes, it will have a greater impact on the calculation of the expected bonus.	Low	Discussed
Code Revising	4.3.6 In Nest_3_MiningContract : oreDrawing() , there are redundant calculations for multiple transactions in the same block.	Info	Fixed
Code Revising	4.3.7 In Nest_3_MiningContract, `_blockMining` has never been initialized and updated.	Info	Fixed
Logical Implementation	4.3.8 After the Nest_NToken_OfferMain contract runs for four years, the mining function fails.	Medium	Fixed
Implementation Flaw	4.3.9 In Nest_3_OfferMain and Nest_NToken_OfferMain, the `turnOut()` function has the risk of re-entrancy attacks.	High	Fixed
Logical Implementation	4.3.10 In the Nest_NToken_OfferMain contract, the offer() function can initially determine whether the transaction's ERC20 Token has a corresponding NToken.	Info	Fixed
Logical Implementation	4.3.11 In the Nest_NToken_TokenAuction contract, the listing operation can be performed without initiating an auction.	Medium	Fixed
Implementation Flaw	4.3.12 NToken `transferFromForOfferPrice()` function has problem of permission verification.	High	Fixed
Logical Implementation	4.3.13 Compatibility is not considered in the code relating to transfers of token.	Low	Fixed

2. Project Information

This part describes the essential information and code structure.

2.1 Essential information

The essential information about NEST Protocol (V3) is shown below:

- Project website
  - https://nestprotocol.org/
- Design details of the protocol
  - https://nestprotocol.org/assets/pdf/ennestwhitepaper.pdf
- Smart contract code
  - Provided by NEST Core Team

2.2 Contract List

The following content shows the main contracts included in Nest Protocol (V3) project:

Contract	Description
IBNEST	NEST Token contract
SuperMan	NestNode Token contract
Nest_3_Abonus	ETH bonus pool contract
Nest_3_Leveling	Leveling contract
Nest_3_NestAbonus	Bonus logic contract
Nest_3_NestSave	NEST save contract
Nest_3_NTokenAbonus	NToken bonus pool
Nest_3_MiningContract	Mining pool contract
Nest_3_OfferMain	Offering contract
Nest_3_OfferPrice	Pricing contract

Nest_3_OfferPriceAdmin	Offering management contract
NEST_NodeAssignment	NestNode bonus assignment contract
NEST_NodeAssignmentData	NestNode bonus data contract
NEST_NodeSave	NestNode bonus save contract
Nest_NToken_TokenAuction	NToken auction contract
Nest_NToken_TokenMapping	NToken mapping contract
Nest_NToken_OfferMain	NToken Offering contract
Nest_NToken_OfferPrice	NToken Pricing contract
Nest_3_VoteFactory	Nest Vote Factory contract

2.3 Audit Record

1. 2020/04/28 The audit begins
2. 2020/05/17 The first submission of the audit issue list
3. 2020/06/02 The second submission of the audit issue list
4. 2020/06/28 The third submission of the audit issue list
5. 2020/07/12 The submission of the final audit report

NEST Protocol(V3) has undergone many changes in design and implementation according to the audit suggestions and own needs during the audit. The report records the main points of the audit process; part of the content may not correspond to the latest code.

3. Code Analysis

This part describes code assessment details, including two items: "role classification" and "functional analysis".

3.1 Role Classification

There are several key roles in the protocol, namely Protocol Owner, NestNode, Offerer, NToken Owner, and Common Account.

- Protocol Owner
  - Description Owner of NEST Protocol
  - Authority
    - The administrator of all contracts in NEST Protocol
    - Add or delete administrator accounts
    - Set the address of protocol mapping contract
    - Set the parameters of protocol contracts
    - Manage the block-list of pricing contract
  - Method of Authorization The creator of the contract, authorized by the administrator or temporarily authorized by Nest voting contract
- NestNode
  - Description Holders of NestNode Token also called NestNode
  - Authority
    - Transfer NestNode Tokens
    - Obtain bonus of NEST Token from offerings
    - Create and participate in NEST vote contracts
    - Trigger the protocol into an emergency state
  - Method of Authorization Holders of NestNode Token
- Offerer
  - Description Participants who provide prices include Offering Miners and Validators. Offering Miner: An account that actively initiates an offer, and provides a price for the NEST price oracle by pledged two-way assets. Validator: An account that has been forced to create a new offer after validating an offer.
  - Authority
    - Obtain revenue from offering mining
    - Obtain revenue when provided prices are queried

        - Retrieve balances from offer when the offer expires
    - Method of Authorization Actively initiate an offer, or create offer when validating an offer
- NToken Owner
    - Description Owners of NToken contracts
    - Authority
        - Obtain revenue from NToken offering mining
        - Transfer ownership of NToken
    - Method of Authorization The creator of NToken, or authorized by transferring the ownership of the contract
- Common Account
    - Description The accounts holding NEST Tokens
    - Authority
        - Transfer tokens in its account
        - Approve other accounts to transfer
        - Lock NEST Tokens to vote in NEST vote contract
        - Lock NEST Tokens to obtain bonus
        - Create and participate in NToken Auctions
    - Method of Authorization Holders of NEST Token

3.2 Functional Analysis

NEST Protocol (V3) contract is a distributed price oracle network protocol. In the contract, the offering miner actively initiates an offer to obtain NEST Token and NToken as mining revenue. The mining revenue of NEST Token and NToken is distributed in each block, and each offering miner shares the mining revenue in the same block. Each offer requires pledging two assets corresponding to the offer price. Other accounts can validate and take orders. The validator exchanges the asset for the offer price. When taking the order, the validator needs to re-offer and pledge the two assets according to the deal price. When the price deviates from the last price, the validator needs to increase the pledged assets, thereby reducing the risk of malicious offers.

The offer has a certain life cycle. In the life cycle of the offer, new offers will be generated after validation, and the price of the corresponding asset pair will be updated at the same time, which is closer to the actual price. NEST Protocol (V3) contract counts the asset pairs' prices corresponding to the offers that are hung up and completed within a certain period of time to form an on-chain price oracle.

Offering miners obtain NEST Token through offering mining. NEST Token holders can choose to lock NEST Tokens to earn bonuses and participate in voting. NEST Token holders can also create and join in the NToken auction, and the successful auctioneer becomes the owner of the NToken contract and obtains the benefits of NToken mining.

NestNode holds the NestNode Token and can regularly receive the NEST Token bonus. NestNode can create NEST votes, for Nest Token holders to decide whether the contract is upgraded.

NEST Protocol (V3) contract has a bonus system that incentivizes NEST Token holders to lock NEST Tokens and get regular bonuses.

We can divide the critical functions of the NEST Protocol (V3) into several parts:

- Offering miners' mining of NEST and NToken

  Offering miners invoke the `offer()` function in the Nest_3_OfferMain and Nest_NToken_OfferPrice contracts to create offers to participate in mining. The `createOffer()` function generates an offer, and the `oreDrawing()` and `mining()` functions are used for the calculation and distribution of the mining tokens.

- Validator's order-taking operation for NEST and NToken offers

  The functions of the NEST and NToken offering to implement the order operation are the `sendEthBuyErc()` and `sendErcBuyEth()` functions in the Nest_3_OfferMain and Nest_NToken_OfferMain contracts. The `createOffer()` function pledges assets and generates new offers.

- Price checking function

  NEST Token holders call the `activation()` function to activate price checking authority. Activated users can call the `updateAndCheckPriceNow()` and `updateAndCheckPriceList()` functions in the Nest_3_OfferPrice contract, and pay ETH to check prices.

4. Audit Detail

This part describes the process and detailed audit results and demonstrates the problems and potential risks.

4.1 Audit Process

The audit strictly followed the audit specification of SECBIT Labs. We analyzed the project from code bug, logical implementation, and potential risks. The process consists of four steps:

- Thoroughly analysis of code line by line.
- Evaluation of vulnerabilities and possible risks revealed in the source code.
- Communication on assessment and confirmation.
- Audit report writing.

4.2 Audit Result

After scanning with adelaide, sf-checker, and badmsg.sender (internal version) developed by SECBIT Labs and open source tools including Mythril, Slither, SmartCheck, and Securify, the auditing team performed a manual assessment. The team inspected the contract line by line, and the result could be categorized into twenty-one types:

Number	Classification	Result
1	Normal functioning of features defined by the contract	✓
2	No obvious bug (e.g., overflow, underflow)	✓
3	Pass Solidity compiler check with no potential error	✓
4	Pass common tools check with no obvious vulnerability	✓
5	No obvious gas-consuming operation	!
6	Meet with ERC20	✓
7	No risk in low-level call (call, delegatecall, callcode) and in-line assembly	✓
8	No deprecated or outdated usage	✓
9	Explicit implementation, visibility, variable type and Solidity version number	✓
10	No redundant code	!
11	No potential risk manipulated by timestamp and network environment	✓
12	Explicit business logic	✓
13	Implementation consistent with annotation and other info	✓
14	No hidden code about any logic that is not mentioned in design	✓

15	No ambiguous logic	✓
16	No risk threatening the developing team	✓
17	No risk threatening exchanges, wallets, and DApps	✓
18	No risk threatening token holders	✓
19	No privilege on managing others' balances	✓
20	No minting method	✓
21	Correct managing hierarchy	✓

4.3 Issues

4.3.1 In Nest_3_NestAbonus : reloadTimeAndMapping(), `_snapshot` cannot provide the function of judging whether the snapshot is completed.

Risk Type	Risk Level	Impact	Status
Logical Implementation	Info	Function failure	Fixed

Description

In `Nest_3_NestAbonus`, `reloadTimeAndMapping()` function uses `_snapshot` to judge whether the snapshot is complete. However, `_snapshot` is not updated after the snapshot; thus, every judgment returns `false`.

```
 if (_snapshot[token][_times.sub(1)] == false) {
            if (token == address(0x0)) {
                _abonusMapping[address(0x0)] = 
_abonusContract.getETHNum();
              //  快照 eth 数量
            } else {
                _abonusMapping[address(token)] = 
ERC20(address(token)).balanceOf(address(_NTokenAbonus));
            //  快照 token数量
            }
        }
```

Suggestion

It is suggested that update`_snapshot[token][_times.sub(1)]` as `true` after each snapshot.

Status

It is fixed according to the suggestion.

4.3.2 In Nest_3_VoteFactory : changeStateOfEmergency(), the implementation logic of emergency state modification is in doubt.

Risk Type	Risk Level	Impact	Status
Logical Implementation	Low	Implementation logic	Fixed

Description

```
    function changeStateOfEmergency() public {
        if (_stateOfEmergency == false) {
            require(_NNToken.balanceOf(address(this)) ==
_emergencyNNAmount);
            _stateOfEmergency = true;
            _emergencyTime = now;
        } else {
            if (_NNToken.balanceOf(address(this)) ==
_emergencyNNAmount || now > _emergencyTime.add(_emergencyTimeLimit)) {
                _stateOfEmergency = false;
                _emergencyTime = 0;
            }
        }
    }
```

In `Nest_3_VoteFactory`, the modification of emergency state is through`changeStateOfEmergency()`function, where the emergency state could be switched by any user when the condition`_NNToken.balanceOf(address(this)) == _emergencyNNAmount` is met. While when the locked NestNode Token is larger than `_emergencyNNAmount`, the condition is not met.

Suggestion

It is recommended to check whether the user calling the function has locked NestNode， that is, add the condition`require(_emergencyPerson[address(tx.origin)] > 0);` and modify the condition on the amount of NestNode Token to `_NNToken.balanceOf(address(this)) >= _emergencyNNAmount`.

Status

It is fixed according to the suggestion.

4.3.3 In Nest_3_VoteContract : constructor(), when the voting contract is in an emergency state, the calculation of the number of total votes and passed votes does not match the actual one.

Risk Type	Risk Level	Impact	Status
Logical Implementation	Low	Affect the transaction price within a limited range	Fixed

Description

When the voting contract is in an emergency state, NestNode does not participate in the voting. `_totalAmount` is set to the initial value, and all NestNode are set to voted pass in default, `_circulation`represents the number of votes passed by the current vote. In the calculation of `_totalAmount` and `_circulation`, `_nestAbonus.checkAllValueMapping(2)` is a snapshot of Nest Token's circulation, which is calculated by `Nest_3_NestAbonus:allValue()`. Therefore, in an emergency state, the calculation of `_circulation` and `_totalAmount` does not match the actual one.

```
        constructor (address contractAddress) public {
        //  初始化
        _voteFactory = Nest_3_VoteFactory(address(msg.sender));
        _nestToken = ERC20(_voteFactory.checkAddress("nest"));
        _NNToken = ERC20(_voteFactory.checkAddress("nestNode"));
        _implementContract =
Nest_3_Implement(address(contractAddress));
        _implementAddress = address(contractAddress);
        _destructionAddress =
address(_voteFactory.checkAddress("nest.v3.destruction"));
        _creator = address(tx.origin);
        _NNUsedCreate = _voteFactory.checkNNUsedCreate();
        _createTime = now;

        _endTime = _createTime.add(_voteFactory.checkLimitTime());
        if (_voteFactory.checkStateOfEmergency() == false) {
            _miningSave =
Nest_3_MiningSave(_voteFactory.checkAddress("nest.v3.miningSave"));
            _nestSave =
Nest_3_NestSave(_voteFactory.checkAddress("nest.v3.nestSave"));
            _circulation = (uint256(10000000000
ether).sub(_nestToken.balanceOf(address(_miningSave))).sub(_nestToken.
balanceOf(address(_destructionAddress)))).mul(_voteFactory.checkCircul
ationProportion()).div(100);
            uint256 NNProportion = _voteFactory.checkNNProportion();
            _NNSingleVote = (uint256(10000000000
ether).sub(_nestToken.balanceOf(address(_miningSave)))).mul(NNProporti
on).div(150000);
        } else {
            _nestAbonus =
Nest_3_NestAbonus(_voteFactory.checkAddress("nest.v3.nestAbonus"));
            _circulation = (uint256(10000000000
ether).sub(_nestAbonus.checkAllValueMapping(2)).sub(_nestToken.balance
Of(address(_destructionAddress)))).mul(_voteFactory.checkCirculationPr
oportion()).div(100);
            uint256 NNProportion = _voteFactory.checkNNProportion();
            _totalAmount = (uint256(10000000000
ether).sub(_nestAbonus.checkAllValueMapping(2))).mul(NNProportion).div
(100);
```

```
        }
    }
```

```
function allValue() public view returns (uint256) {
    uint256 all = 10000000000 ether;
    uint256 leftNum =
all.sub(_nestContract.balanceOf(address(_voteFactory.checkAddress("nest.v3.miningSave")))).sub(_nestContract.balanceOf(address(_destructionAddress)));
    return leftNum;
}
```

Suggestion

It is recommended to change the calculation of `_circulation` and `_totalAmount` to`_circulation = _nestAbonus.checkAllValueMapping(2).mul(_voteFactory.checkCirculationProportion()).div(100)`与`_totalAmount = _nestAbonus.checkAllValueMapping(2).add(_nestToken.balanceOf(address(_destructionAddress))).mul(NNProportion).div(100)`.

Status

The voting logic in the emergency state has been modified, and the problem has been fixed.

4.3.4 In Nest_3_OfferPrice and Nest_NToken_OfferPrice contracts, the update of `blockAddress` in the contract is inconsistent with the contract design document.

Risk Type	Risk Level	Impact	Status
Logical Implementation	Info	Inconsistent documentation and implementation	Fixed

Description

In the contract design document, `blockAddress` is used to record the first offering miner in the block. In the code implementation of the `addPrice()` function, `blockAddress` records the last offering miner in the block.

```
    function addPrice(uint256 ethAmount, uint256 tokenAmount, uint256 
endBlock, address tokenAddress) public onlyFactory {
    ......
        //  更新最后一个报价矿工
        blockAddress[block.number][tokenAddress] = address(tx.origin);
    }
```

Suggestion

It is recommended to determine the meaning of `blockAddress` and modify the code or design document.

Status

It has been determined that `blockAddress` represents the last offering miner.

4.3.5 In Nest_3_NestAbonus contract, when the expected bonus increment ratio changes, it will have a greater impact on the calculation of the expected bonus.

Risk Type	Risk Level	Impact	Status
Logical Implementation	Low	Implementation Logic	Discussed

Description

In the NEST3.0 economic model, for every increase in the NEST circulation of 100 million, the ETH number of the minimum bonus is expected to increase by a certain percentage, and its quantity change should be continuous. The expected minimum bonus is calculated in the `levelingResult()` function, and the change in the bonus ratio is calculated from the time when the NEST circulation is 0, that is

$$(1 + expectedIncrement)^{nestAllValue/100000000}$$

Therefore, when the expected bonus increase ratio changes, the calculation of the amount of the minimum bonus ETH in the current period will have an exponential effect.

```
uint256 miningAmount = allValue().div(100000000 ether);
        uint256 minimumAbonus = _expectedMinimum;
        for (uint256 i = 0; i < miningAmount; i++) {
            minimumAbonus = 
minimumAbonus.add(minimumAbonus.mul(_expectedIncrement).div(100));
        }
```

Suggestion

It is recommended to modify the calculation method of the expected minimum bonus (such as updating the storage one by one) to reduce the impact of parameter changes on the calculation of the current minimum bonus.

Status

It has been confirmed to keep the existing logic.

4.3.6 In Nest_3_MiningContract： oreDrawing() , there are redundant calculations for multiple transactions in the same block.

Risk Type	Risk Level	Impact	Status
Code Revising	Info	Redundant code consumes unnecessary gas	Fixed

Description

In the NEST protocol, users mine at the same time as the offering. The first offering transaction in the same block can mine NEST Tokens from the mining pool contract. The number of mining is calculated by the `changeBlockAmountList()` function. The first call in the block returns the number of mined tokens, `_latestMining` is updated to `block.number`, and then when the function is called in the same block, the return value is 0. Therefore, when the mining quantity is 0, `_nestContract.transfer(address(msg.sender), miningAmount);` and `emit OreDrawingLog(block.number,miningAmount,msg.value);` in `oreDrawing()` are both redundant operations.

```
    function oreDrawing() public payable returns (uint256) {
        require(address(msg.sender) == _offerFactoryAddress, "No
authority");
        //  更新出矿量列表
        uint256 miningAmount = changeBlockAmountList();
        //  转手续费
        repayEth(msg.value);
        //  转 nest
        _nestContract.transfer(address(msg.sender), miningAmount);

        emit OreDrawingLog(block.number,miningAmount,msg.value);
        return miningAmount;
    }
```

Suggestion

It is recommended that in the implementation of `oreDrawing()`, first check whether the number of mining is 0 by `_latestMining` equal to `block.number`, and then execute the relevant code.

Status

It is fixed according to the suggestion.

4.3.7 In Nest_3_MiningContract, `_blockMining` has never been initialized and updated.

Risk Type	Risk Level	Impact	Status
Code Revising	Info	There is an unused mapping	Fixed

Description

In Nest_3_MiningContract, `_blockMining` is used to store the mining amount corresponding to the block number. This mapping is not updated in the code.

Suggestion

It is recommended to update `_blockMining` after effective mining in the implementation of `oreDrawing()`.

Status

The `_blockMining` variable has been deleted, and the problem has been fixed.

4.3.8 After the Nest_NToken_OfferMain contract runs for four years, the mining function fails.

Risk Type	Risk Level	Impact	Status
Logical Implementation	Medium	NToken mining function failure	Fixed

Description

In Nest_NToken_OfferMain, the `_attenuationAmount` array is used to store the number of mining at different time stages, and the array length is 4. Therefore, when the contract runs to the fifth year, when the `oreDrawing()` function accesses `_attenuationAmount[4]`, the array crosses the boundary, resulting in an error in the function execution and NToken mining function failure.

```
    function oreDrawing(address NToken) private returns(uint256) {
        Nest_NToken miningToken = Nest_NToken(NToken);
        (uint256 createBlock, uint256 recentlyUsedBlock) =
miningToken.checkBlockInfo();
        uint256 attenuationPointOld =
recentlyUsedBlock.sub(createBlock).div(_blockAttenuation);
        uint256 attenuationPointNow =
block.number.sub(createBlock).div(_blockAttenuation);
        uint256 miningAmount;
        uint256 attenuationPointend =
createBlock.add(attenuationPointOld.add(1).mul(_blockAttenuation));
            for (uint256 i = attenuationPointOld; i <
attenuationPointNow; i++) {
                miningAmount =
miningAmount.add(attenuationPointend.sub(recentlyUsedBlock).mul(_atten
uationAmount[i]));
                recentlyUsedBlock = attenuationPointend;
                attenuationPointend =
attenuationPointend.add(_blockAttenuation);
            }
        miningAmount =
miningAmount.add(_attenuationAmount[attenuationPointNow].mul(block.num
ber.sub(recentlyUsedBlock)));
        miningToken.increaseTotal(miningAmount);
        emit OreDrawingLog(block.number, miningAmount, NToken);

        return miningAmount;
    }
```

Suggestion

It is recommended to increase the judgment of array subscripts in the implementation of `oreDrawing()`. If the subscript is greater than 3, the value of `_attenuationAmount[3]` is returned.

Status

Fixed.

4.3.9 In Nest_3_OfferMain and Nest_NToken_OfferMain, the `turnOut()` function has the risk of re-entrancy attacks.

Risk Type	Risk Level	Impact	Status
Implementation Flaw	Low	Violation of secure development regulations	Fixed

Description

In Nest_3_OfferMain and Nest_NToken_OfferMain, the turnOut() function executes `repayEth(offerPriceData.owner, offerPriceData.ethAmount)`. If `offerPriceData.owner` is a contract account, the contract will be able to call `turnOut()` in the fallback function after receiving the transfer and initiate a re-entrancy attack.

Although the `offer()` function restricts the calling account from being a contract account, `sendEthBuyErc()` and `sendErcBuyEth()` do not limit the account, and then initial the offer with `createOffer()`. Therefore, the contract account initiates the offers by calling these two functions and can initiate a re-entrancy attack after the offer expires.

Since `transfer()` is used in the `repayEth()` function for Ethereum transfers, the call is fixed to 2300 gas, so it cannot currently cause substantial harm.

However, the specific implementation of `ERC20(address(offerPriceData.tokenAddress)).safeTransfer()` is unknown and unpredictable, so there are still certain risks. It is recommended to follow the smart contract security development specification here, and adopt the paradigm of Checks-Effects-Interactions, first Verify, then modify the key parameters, and finally carry out external interaction.

```
    function turnOut(address contractAddress) public {
        uint256 index = toIndex(contractAddress);
        Nest_3_OfferPriceData storage offerPriceData = _prices[index];
        require(checkContractState(offerPriceData.blockNum) == 1,
"Offer status error");

        // 取出ETH
        if (offerPriceData.ethAmount > 0) {
            repayEth(offerPriceData.owner, offerPriceData.ethAmount);
            offerPriceData.ethAmount = 0;
        }

        // 取出ERC20
        if (offerPriceData.tokenAmount > 0) {

 ERC20(address(offerPriceData.tokenAddress)).safeTransfer(offerPriceDa
ta.owner, offerPriceData.tokenAmount);
            offerPriceData.tokenAmount = 0;
        }
        // 挖矿结算
        if (offerPriceData.serviceCharge > 0) {
            uint256 myMiningAmount =
offerPriceData.serviceCharge.mul(_offerBlockMining[offerPriceData.bloc
kNum]).div(_offerBlockEth[offerPriceData.blockNum]);
```

```
            _nestToken.safeTransfer(offerPriceData.owner, 
myMiningAmount);
            offerPriceData.serviceCharge = 0;
        }

    }
```

```
    function turnOut(address contractAddress) public {
        uint256 index = toIndex(contractAddress);
        Nest_NToken_OfferPriceData storage offerPriceData = 
_prices[index];
        require(checkContractState(offerPriceData.blockNum) == 1, 
"Offer status error");
        // 取出ETH
        if (offerPriceData.ethAmount > 0) {
            repayEth(offerPriceData.owner, offerPriceData.ethAmount);
            offerPriceData.ethAmount = 0;
        }
        // 取出ERC20
        if (offerPriceData.tokenAmount > 0) {

 ERC20(address(offerPriceData.tokenAddress)).transfer(offerPriceData.o
wner, offerPriceData.tokenAmount);
            offerPriceData.tokenAmount = 0;
        }
        // 挖矿结算
        if (offerPriceData.mining) {
            mining(offerPriceData.blockNum, 
offerPriceData.tokenAddress);
            offerPriceData.mining = false;
        }
    }
```

Suggestion

It is recommended to restrict the calling account in `sendEthBuyErc()` and `sendErcBuyEth()` and execute `offerPriceData.ethAmount = 0` before calling `repayEth()`. You may also consider using [ReentrancyGuard]() to protect all necessary functions from reentry.

Status

Fixed.

4.3.10 In the Nest_NToken_OfferMain contract, the offer() function can initially determine whether the transaction's ERC20 Token has a corresponding NToken.

Risk Type	Risk Level	Impact	Status
Code Revising	Info	Function implementation logic is unreasonable	Fixed

Description

In the Nest_NToken_OfferMain contract, the `offer()` function judges the validity of NToken only after performing operations such as price comparison, offer creation, and token transfer.

Suggestion

It is recommended to add a judgment on the mapping of `erc20Address` to NToken at the beginning of the `offer()` function.

Status

Fixed

4.3.11 In the Nest_NToken_TokenAuction contract, the listing operation can be performed without initiating an auction.

Risk Type	Risk Level	Impact	Status
Implementation Flaw	Medium	NToken creation mechanism fails	Fixed

Description

In the Nest_NToken_TokenAuction contract, since the value of the `_auctionList[token]` structure is not initialized, the conditions of `now> _auctionList[token].endTime` and `_nestToken.transfer(_destructionAddress, _auctionList[token].auctionValue)` can be satisfied That is, the `auctionSuccess()` function can be executed normally when the auction is not initiated, thereby storing an invalid NToken mapping. Users can successfully auction and create NToken contracts without Nest Token, which leads to the failure of the NToken creation mechanism. The administrator needs to reset the NToken mapping in the Nest_NToken_TokenMapping contract to return to normal.

```
function auctionSuccess(address token) public {
        Nest_3_NestAbonus nestAbonus =
Nest_3_NestAbonus(_voteFactory.checkAddress("nest.v3.nestAbonus"));
        uint256 nowTime = now;
        uint256 nextTime = nestAbonus.getNextTime();
```

```
        uint256 timeLimit = nestAbonus.checkTimeLimit();
        uint256 getAbonusTimeLimit =
nestAbonus.checkGetAbonusTimeLimit();
        require(!(nowTime >= nextTime.sub(timeLimit) && nowTime <=
nextTime.sub(timeLimit).add(getAbonusTimeLimit)), "Not time to
auctionSuccess");
        require(now > _auctionList[token].endTime, "Token is on
sale");
        //  初始化NToken
        Nest_NToken NToken = new Nest_NToken(strConcat("NToken",
getAddressStr(token)), strConcat("N", getAddressStr(token)),
address(_voteFactory), _createValue, _NTokenAbonusValue,
_NTokenAbonus, address(_auctionList[token].latestAddress));
        //  拍卖资金销毁
        require(_nestToken.transfer(_destructionAddress,
_auctionList[token].auctionValue), "Transfer failure");
        //  加入 NToken映射
        _tokenMapping.addTokenMapping(token, address(NToken));
    }
```

Suggestion

It is recommended to add a judgment on the auction initiation state in the `auctionSuccess()` function.

Status

Fixed.

4.3.12 NToken `transferFromForOfferPrice()` function has problem of permission verification.

Risk Type	Risk Level	Impact	Status
Implementation Flaw	High	Permission verification fails	Fixed

Description

```
function transferFromForOfferPrice(address from, address to, uint256
value) public returns (bool) {
 _transfer(from, to, value);
 emit Approval(from, msg.sender, _allowed[from][msg.sender]);
 return true;
}
```

Suggestion

Remove the function or add proper permission verification.

Status

Fixed. The function has been removed.

4.3.13 Compatibility is not considered in the code relating to transfers of token.

Risk Type	Risk Level	Impact	Status
Logical Implementation	Low	Compatibility of token transfers	Fixed

Description

```
function startAnAuction(address token, uint256 auctionAmount) public {
 require(_tokenMapping.checkTokenMapping(token) == address(0x0),
"Token already exists");
 require(_auctionList[token].endTime == 0, "Token is on sale");
 require(auctionAmount >= _minimumNest, "AuctionAmount less than
the minimum auction amount");
 require(_nestToken.transferFrom(address(msg.sender),
address(this), auctionAmount), "Authorization failed");
 require(_tokenBlackList[token] == false);
 AuctionInfo memory thisAuction = AuctionInfo(now.add(_duration),
auctionAmount, address(msg.sender), auctionAmount);
 _auctionList[token] = thisAuction;
 _allAuction.push(token);
}
```

The implementation of some tokens does not strictly follow the interface definition of ERC20; thus, compatibility problems may occur when interacting with these tokens and affect the protocol's normal operation.

Suggestion

Use `safeTransfer()` and `safeTransferFrom()` functions to process the calls to tokens.

Status

Fixed.

4.4 Risks

4.4.1 When the NEST protocol bonus pool is small, the bonus mechanism may be invalidated because the transaction gas consumption is greater than the revenue.

Risk Type	Risk Level	Impact	Status
Potential Risk	Info	Stability of the bonus mechanism	Discussed

Description

The NEST protocol encourages users to lock the NEST Token through the dividend mechanism and receive bonuses regularly. Users with locked positions receive bonus rewards by calling `Nest_3_NestAbonus:getAbonus()`. There are many update and calculation operations in the execution of this function, such as the calculation of `minimumAbonus` in `levelingResult()`, increasing the required gas to initiate the transaction. According to the testnet deployment and contract transactions, calling `getAbonus()` requires about [16w of gas] (https://ropsten.etherscan.io/tx/0xd26498f75c0abc4a8d14859602a6c1c8fa7d2d8541dd256dd8225ab1b8841610), and take the current Ethereum mainnet price 30GWei for calculation, it takes 30GWei*160000≈0.005 Ether to perform a bonus collection operation. Therefore, when the user's revenue reaches 0.005 Ether, the bonus mechanism is effective.

```
    function levelingResult() private {
        uint256 thisAbonus = _abonusContract.getETHNum();
        if (thisAbonus > 10000 ether) {

 _abonusContract.getETH(thisAbonus.mul(_savingLevelThree).div(100),
address(_nestLeveling));
        } else if (thisAbonus > 1000 ether) {

 _abonusContract.getETH(thisAbonus.mul(_savingLevelTwo).div(100),
address(_nestLeveling));
        } else if (thisAbonus > 100 ether) {

 _abonusContract.getETH(thisAbonus.mul(_savingLevelOne).div(100),
address(_nestLeveling));
        }

        uint256 miningAmount = allValue().div(100000000 ether);
        uint256 minimumAbonus = _expectedMinimum;
        for (uint256 i = 0; i < miningAmount; i++) {
```

```
            minimumAbonus =
minimumAbonus.add(minimumAbonus.mul(_expectedIncrement).div(100));
        }
        uint256 nowEth = _abonusContract.getETHNum();
        if (nowEth < minimumAbonus) {
            _nestLeveling.tranEth(minimumAbonus.sub(nowEth),
address(_abonusContract));
        }
    }
```

Suggestion

It is recommended to reduce unnecessary code execution in the extraction of bonuses and reduce gas consumption. Increase the ETH bonus pool scale, increase the user's bonus revenue, and increase the user's enthusiasm for participating in the dividend.

Status

It has been confirmed to keep the existing logic.

4.4.2 NEST Protocol administrator and community governance.

Risk Type	Risk Level	Impact	Status
Protocol Design	Info	Governance security	Discussed

Description

In the design and implementation of the NEST Protocol, the admin interface `onlyOwner` is set for the setup and update of the protocol. It is necessary to keep the private key of administrators safe to guarantee that the protocol cannot be tampered maliciously.

Besides, to avoid the admin account's single failure, the NEST Protocol is planning to remove the admin authority and transform it into a community voting governance model. The voting governance is currently flexible, and long-term voting governance requires NestNode and voters to ensure each vote proposal's security.

Status

Discussed.

5. Conclusion

After auditing and analyzing the contract code of NEST Protocol V3, SECBIT Labs found some issues to optimize and proposed corresponding suggestions, which have been shown above. SECBIT Labs holds that NEST Protocol implements on-chain price oracles with economic incentives and game mechanisms. The protocol design is generally ingenious, which entirely arises the enthusiasm of each role in the protocol and has made unique design and detail optimization from the aspect of on-chain price security.

Disclaimer

SECBIT smart contract audit service assesses the contract's correctness, security, and performability in code quality, logic design, and potential risks. The report is provided "as is", without any warranties about the code practicability, business model, management system's applicability, and anything related to the contract adaptation. This audit report is not to be taken as an endorsement of the platform, team, company, or investment.

APPENDIX

Vulnerability/Risk Level Classification

Level	Description
High	Severely damage the contract's integrity and allow attackers to steal ethers and tokens, or lock ethers inside the contract.
Medium	Damage contract's security under given conditions and cause impairment of benefit for stakeholders.
Low	Cause no actual impairment to contract.
Info	Relevant to practice or rationality could possibly bring risks.

SECBIT Labs is devoted to constructing a common-consensus, reliable and ordered blockchain economic entity.

http://www.secbit.io

audit@secbit.io

[@secbit_io](https://twitter.com/secbit_io)