BEOSIN
Blockchain Security

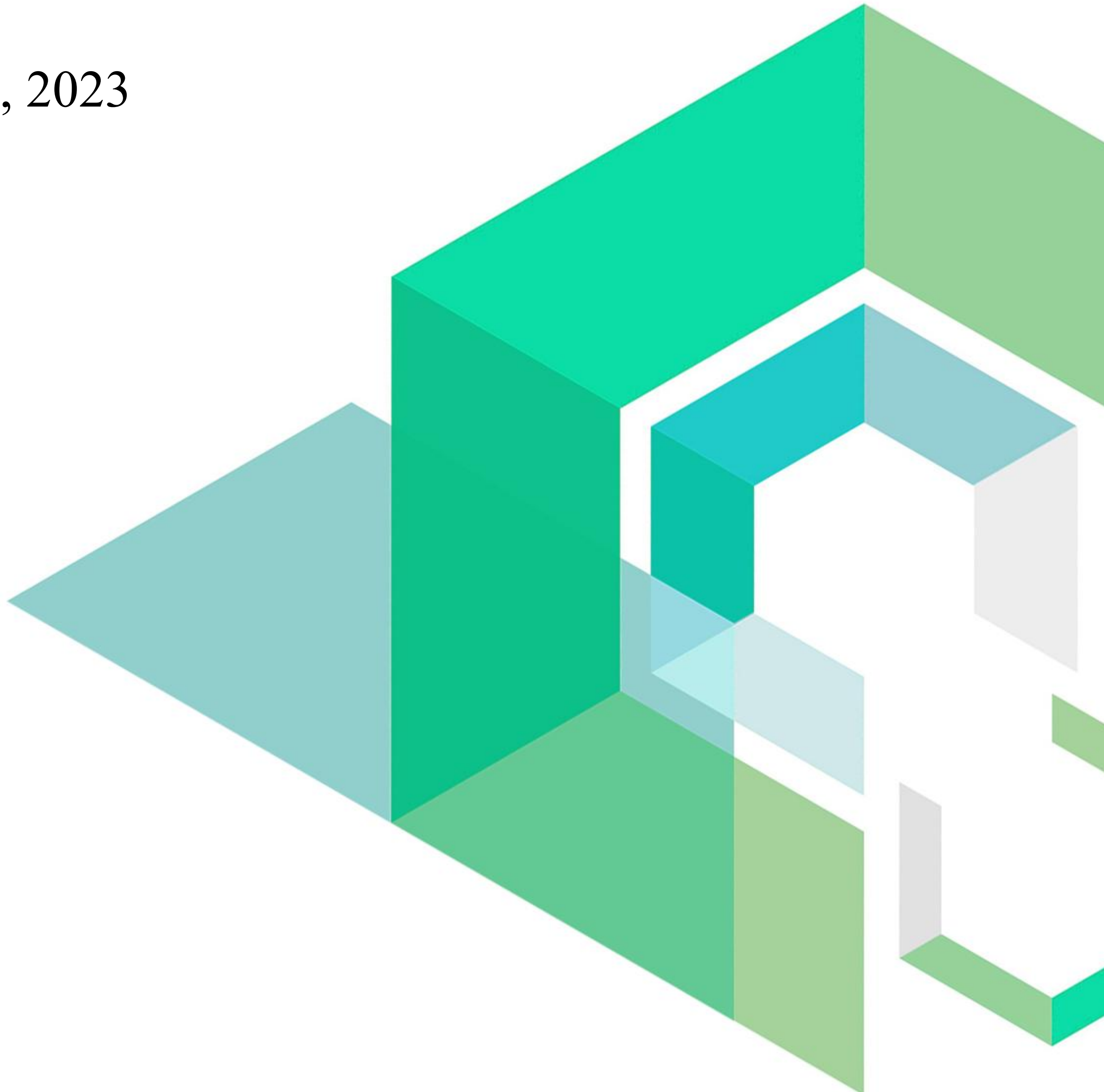
NEWFI

Smart Contract Security Audit

V1.0

No. 202305171748

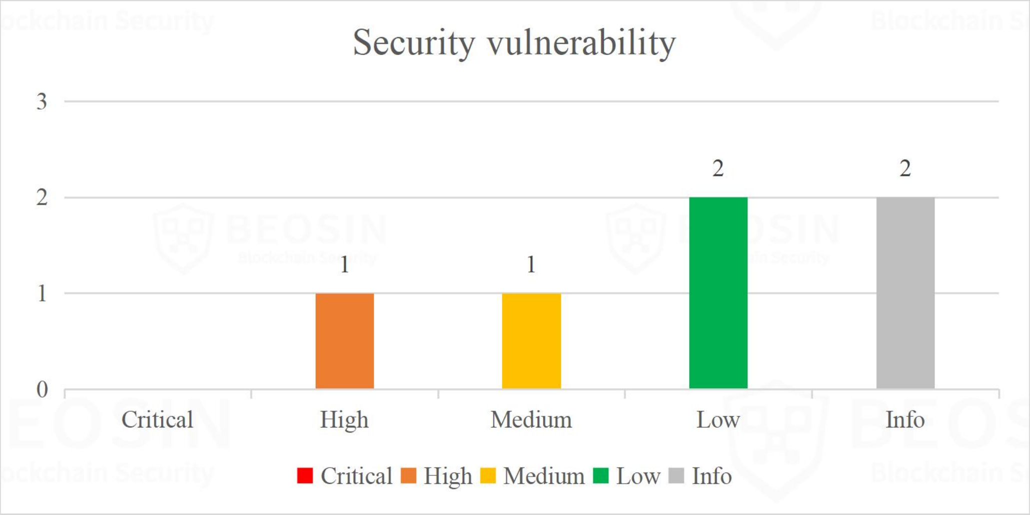
May 17th, 2023



Contents

Summary of Audit Results

After auditing, 1 High-risk and 1 Medium-risk and 2 Low-risk and 2 Info item were identified in the NEWFI project. Specific audit details will be presented in the **Findings** section. Users should pay attention to the following aspects when interacting with this project:



***Notes:**

- **Risk Description:**

1. Owner can change the lprate weight of the pool, users need to pay attention when staking.

2. The contract has centralization risks, the owner's authority is too high, and users need to pay attention to the safety of funds when participating.

3. When a user participates in the project, the contract reward will be sent to the current contract, the contract will not record user credentials, and offline app records are not within the scope of audit.

- **Project Description:**

Business overview

NEWFI is a staking project on Ethereum and Binance Smart Chain (BSC) based on MasterChef V3. The contract utilizes the liquidity mining functionality of Uniswap V3. Currently, the contract owner has the ability to create staking pools, add liquidity, swap tokens, and automatically obtain profits. Users participating in the contract do not receive any token or certificate. All rewards are first sent to the contract itself, and to withdraw funds, a signer signature is required. The owner can withdraw assets using functions (excluding the signature) such as *withdrawFarm* and *withdrawNFT*.

1 Overview

1.1 Project Overview

Project Name	NEWFI
Platform	Ethereum & BNB Chain
Project Link	https://github.com/NEWFI/StakedV3/ https://github.com/NEWFI/Compute/

Commit Hash	Initial:
	540c341c8f3bda94b9c3b9a0e12aeaef60b2a2ca(StakedV3)
	eeab52391957df1da57fd9cd4acfac86a0dba41a(Compute)
	Final:
	b8cd481ede7fb95e1cd054c56184fa77e690e4b9(StakedV3)

1.2 Audit Overview

Audit work duration: May 11, 2023 – May 17, 2023

Audit methods: Formal Verification, Static Analysis, Typical Case Testing and Manual Review.

Audit team: Beosin Security Team.

2 Findings

Index	Risk description	Severity level	Status
NEWFI-1	The owner can extract nft arbitrarily	High	Fixed
NEWFI-2	Owner permission is too high	Medium	Partially Fixed
NEWFI-3	The *poolControl* function setting has no range limit	Low	Partially Fixed
NEWFI-4	The *poolCreat* function is improperly designed	Low	Fixed
NEWFI-5	Missing event trigger	Info	Fixed
NEWFI-6	Compiler is not fixed	Info	Fixed

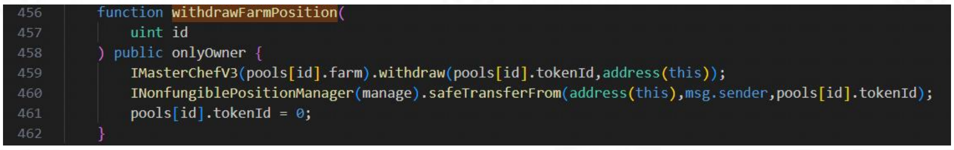
Status Notes:

1. NEWFI-2 is partially fixed and owner can still withdraw funds from the contract through other functions.
2. NEWFI-3 is partially fixed and the setting range does not set a maximum upper limit, the owner can change it.

Finding Details:

[NEWFI-1] The owner can extract nft arbitrarily

<table>
<tr><td><b>Severity Level</b></td><td>High</td></tr>
<tr><td><b>Type</b></td><td>Business Security</td></tr>
<tr><td><b>Lines</b></td><td>StakeV3.sol#L456-462</td></tr>
<tr><td><b>Description</b></td><td>When a user participates in the project, the assets invested through the <i>Invest</i> function will be staked or additionally staked in MasterChef V3, and the initial mint NFT will be used as a shared position and sent to the current contract, and the owner can directly withdraw the ntf of the current contract.



```
456     function withdrawFarmPosition(
457         uint id
458     ) public onlyOwner {
459         IMasterChefV3(pools[id].farm).withdraw(pools[id].tokenId,address(this));
460         INonfungiblePositionManager(manage).safeTransferFrom(address(this),msg.sender,pools[id].tokenId);
461         pools[id].tokenId = 0;
462     }
```

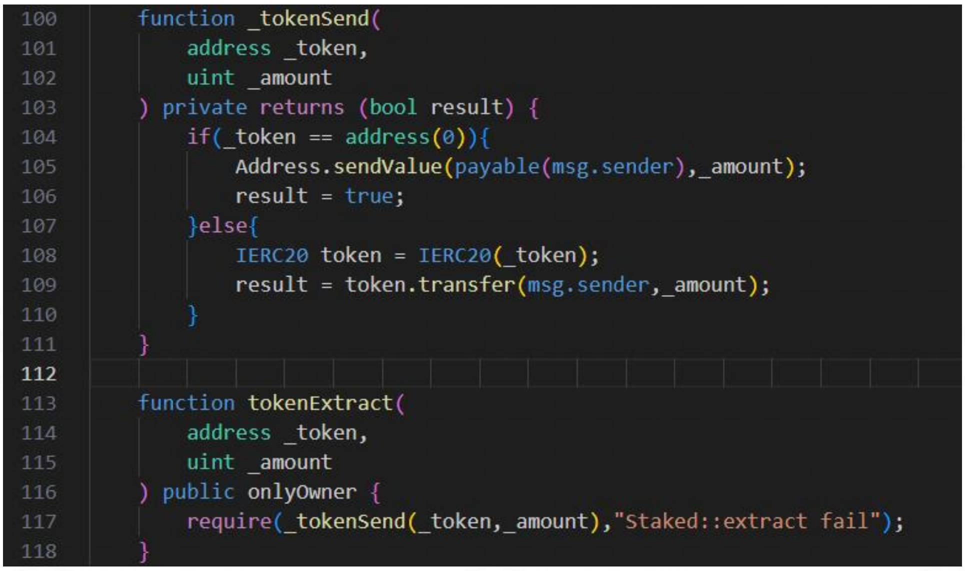
Figure 1 Source code of <i>withdrawFarmPosition</i> function</td></tr>
<tr><td><b>Recommendations</b></td><td>It is recommended to to delete the function.</td></tr>
<tr><td><b>Status</b></td><td>Fixed.</td></tr>
</table>

[NEWFI-2] Owner permission is too high

Severity Level	Medium
Type	Business Security
Lines	StakeV3.sol#L113-118

Description

The owner can directly withdraw the balance of the contract, which may cause certain losses to the user's assets and make the user unable to withdraw.



```
100        function _tokenSend(
101            address _token,
102            uint _amount
103        ) private returns (bool result) {
104            if(_token == address(0)){
105                Address.sendValue(payable(msg.sender),_amount);
106                result = true;
107            }else{
108                IERC20 token = IERC20(_token);
109                result = token.transfer(msg.sender,_amount);
110            }
111        }
112
113        function tokenExtract(
114            address _token,
115            uint _amount
116        ) public onlyOwner {
117            require(_tokenSend(_token,_amount),"Staked::extract fail");
118        }
```

Figure 2 Source code of *tokenExtract* function

Recommendations

It is recommended to ensure that the user's principal can be withdrawn when withdrawing contract assets.

Status

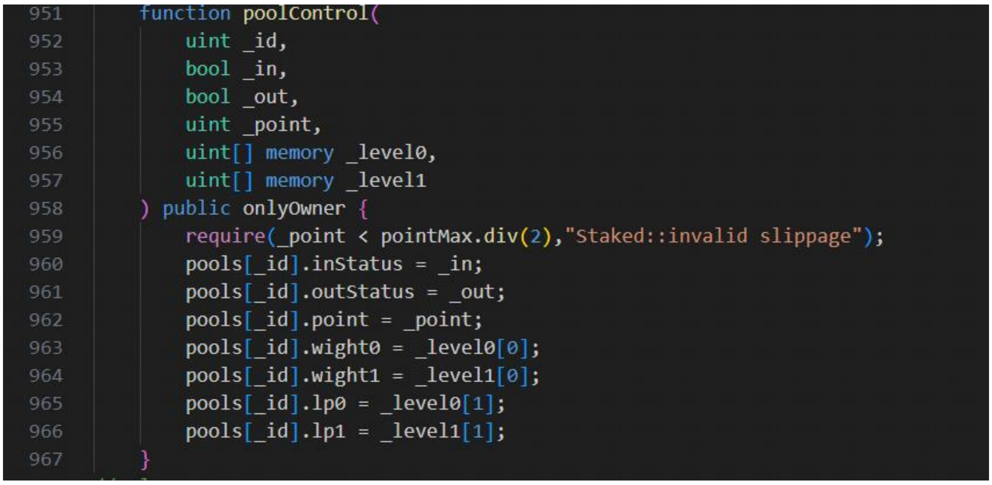
Partially Fixed. The function has been deleted, but the contract owner has high authority and can still withdraw funds through methods such as self-signature and *withdrawNFT* functions, users need to pay attention.

[NEWFI-3] The *poolControl* function setting has no range limit

Severity Level	Low
Type	Business Security
Lines	StakedV3.sol#L951-967

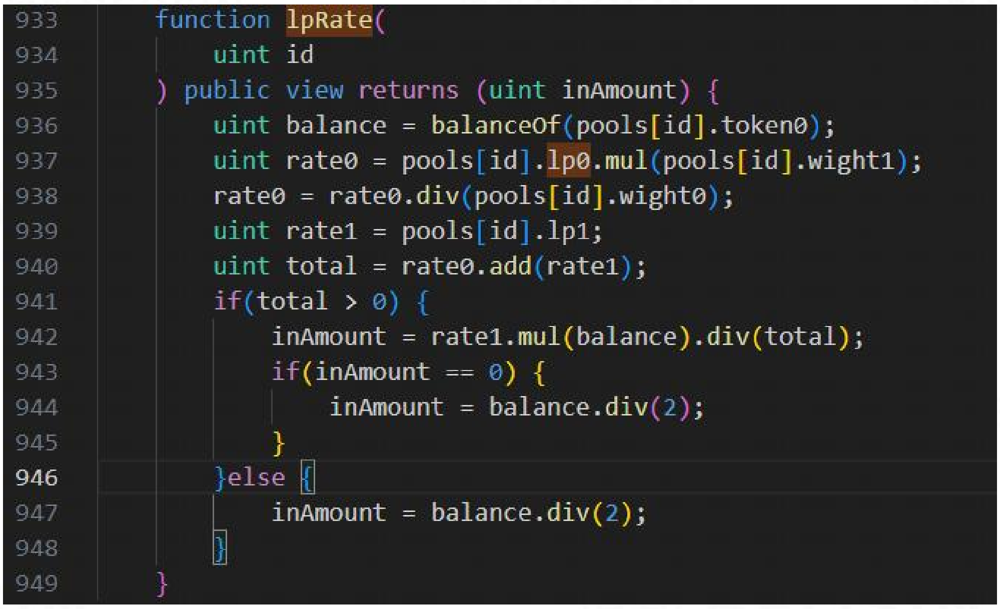
Description

The owner can arbitrarily change the weight of the lprate in the created pool without any scope restrictions, and the owner can control the ratio of stake entry in the current contract.



```
951    function poolControl(
952        uint _id,
953        bool _in,
954        bool _out,
955        uint _point,
956        uint[] memory _level0,
957        uint[] memory _level1
958    ) public onlyOwner {
959        require(_point < pointMax.div(2),"Staked::invalid slippage");
960        pools[_id].inStatus = _in;
961        pools[_id].outStatus = _out;
962        pools[_id].point = _point;
963        pools[_id].wight0 = _level0[0];
964        pools[_id].wight1 = _level1[0];
965        pools[_id].lp0 = _level0[1];
966        pools[_id].lp1 = _level1[1];
967    }
```

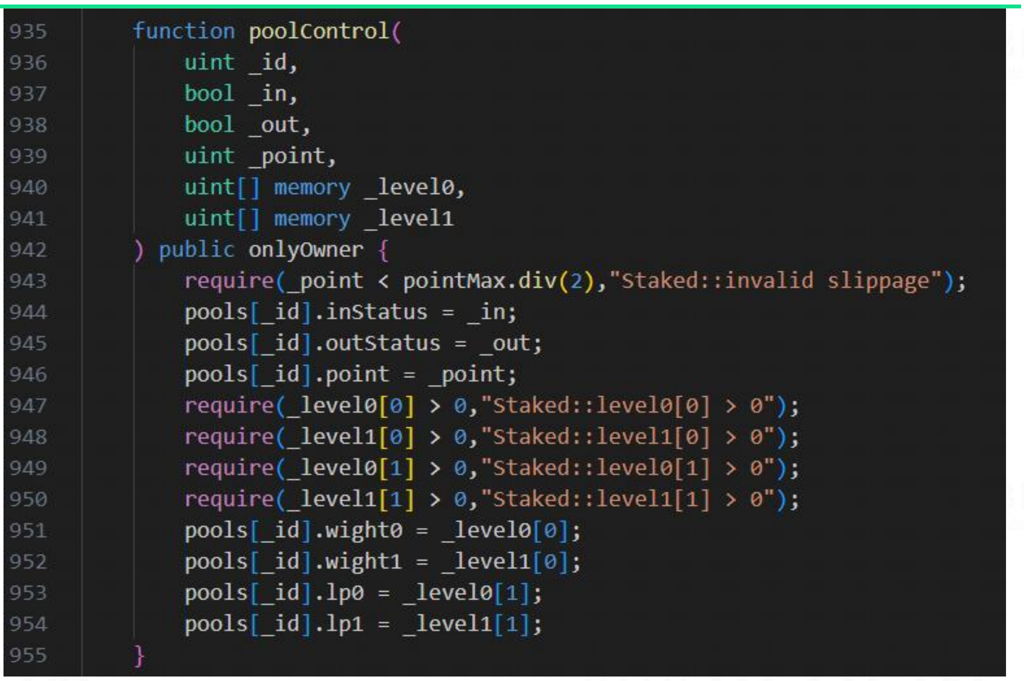
Figure 3 Source code of *poolControl* function (unfixed)



```
933    function lpRate(
934        uint id
935    ) public view returns (uint inAmount) {
936        uint balance = balanceOf(pools[id].token0);
937        uint rate0 = pools[id].lp0.mul(pools[id].wight1);
938        rate0 = rate0.div(pools[id].wight0);
939        uint rate1 = pools[id].lp1;
940        uint total = rate0.add(rate1);
941        if(total > 0) {
942            inAmount = rate1.mul(balance).div(total);
943            if(inAmount == 0) {
944                inAmount = balance.div(2);
945            }
946        }else {
947            inAmount = balance.div(2);
948        }
949    }
```

Figure 4 Source code of *lpRate* function

Recommendations	It is recommended that the owner add scope restrictions when setting.
Status	Partially Fixed.The *poolControl* function does not set a maximum limit.



```
    function poolControl(
        uint _id,
        bool _in,
        bool _out,
        uint _point,
        uint[] memory _level0,
        uint[] memory _level1
    ) public onlyOwner {
        require(_point < pointMax.div(2),"Staked::invalid slippage");
        pools[_id].inStatus = _in;
        pools[_id].outStatus = _out;
        pools[_id].point = _point;
        require(_level0[0] > 0,"Staked::level0[0] > 0");
        require(_level1[0] > 0,"Staked::level1[0] > 0");
        require(_level0[1] > 0,"Staked::level0[1] > 0");
        require(_level1[1] > 0,"Staked::level1[1] > 0");
        pools[_id].wight0 = _level0[0];
        pools[_id].wight1 = _level1[0];
        pools[_id].lp0 = _level0[1];
        pools[_id].lp1 = _level1[1];
    }
```

Figure 5 Source code of *poolControl* function (fixed)

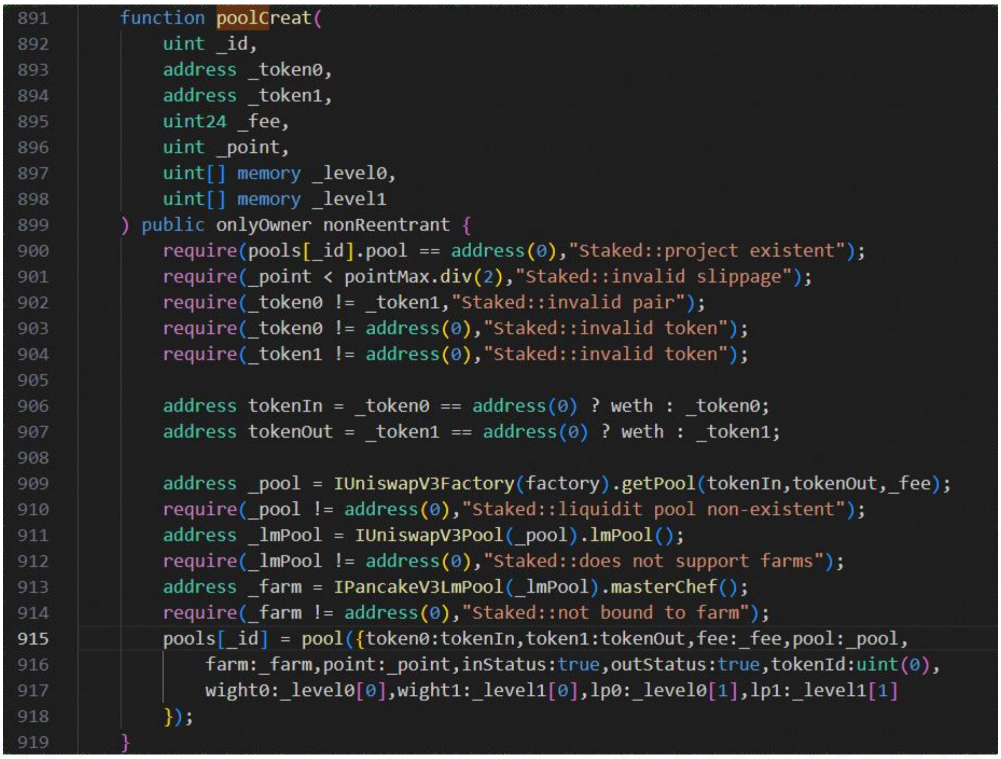
[NEWFI-4] The *poolCreat* function is improperly designed

Severity Level Low

Type Business Security

Lines StakedV3.sol#L891-919

Description When the *poolCreat* function is called, _token0 and _token1 conflict with the judgment of the 0 address, first exclude the 0 address, and then assign a value when it is judged to be the 0 address.

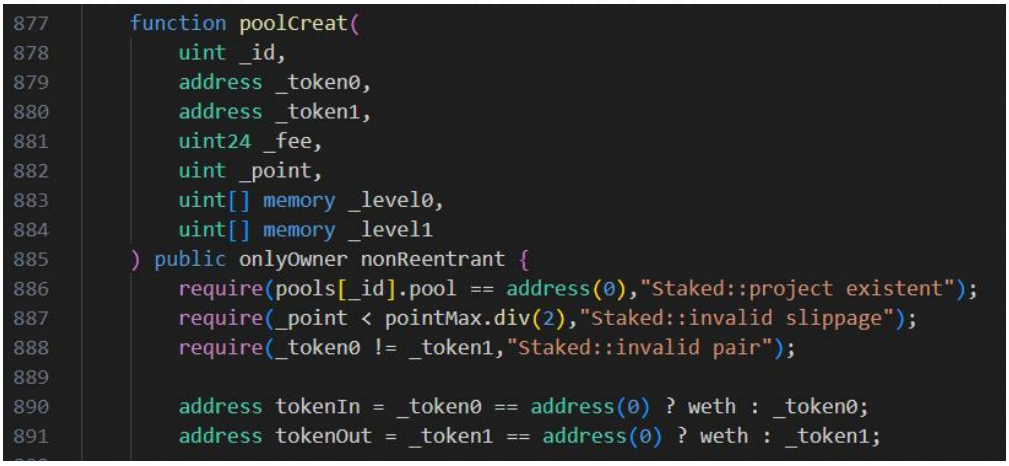


```
891     function poolCreat(
892         uint _id,
893         address _token0,
894         address _token1,
895         uint24 _fee,
896         uint _point,
897         uint[] memory _level0,
898         uint[] memory _level1
899     ) public onlyOwner nonReentrant {
900         require(pools[_id].pool == address(0),"Staked::project existent");
901         require(_point < pointMax.div(2),"Staked::invalid slippage");
902         require(_token0 != _token1,"Staked::invalid pair");
903         require(_token0 != address(0),"Staked::invalid token");
904         require(_token1 != address(0),"Staked::invalid token");
905
906         address tokenIn = _token0 == address(0) ? weth : _token0;
907         address tokenOut = _token1 == address(0) ? weth : _token1;
908
909         address _pool = IUniswapV3Factory(factory).getPool(tokenIn,tokenOut,_fee);
910         require(_pool != address(0),"Staked::liquidit pool non-existent");
911         address _lmPool = IUniswapV3Pool(_pool).lmPool();
912         require(_lmPool != address(0),"Staked::does not support farms");
913         address _farm = IPancakeV3LmPool(_lmPool).masterChef();
914         require(_farm != address(0),"Staked::not bound to farm");
915         pools[_id] = pool({token0:tokenIn,token1:tokenOut,fee:_fee,pool:_pool,
916             farm:_farm,point:_point,inStatus:true,outStatus:true,tokenId:uint(0),
917             wight0:_level0[0],wight1:_level1[0],lp0:_level0[1],lp1:_level1[1]
918         });
919     }
```

Figure 6 Source code of *poolCreat* function (unfixed)

Recommendations It is recommended to delete the require of _token0 and _token1.

Status Fixed.



```
    function poolCreat(
        uint _id,
        address _token0,
        address _token1,
        uint24 _fee,
        uint _point,
        uint[] memory _level0,
        uint[] memory _level1
    ) public onlyOwner nonReentrant {
        require(pools[_id].pool == address(0),"Staked::project existent");
        require(_point < pointMax.div(2),"Staked::invalid slippage");
        require(_token0 != _token1,"Staked::invalid pair");

        address tokenIn = _token0 == address(0) ? weth : _token0;
        address tokenOut = _token1 == address(0) ? weth : _token1;
```

Figure 7 Source code of *poolCreat* function (fixed)

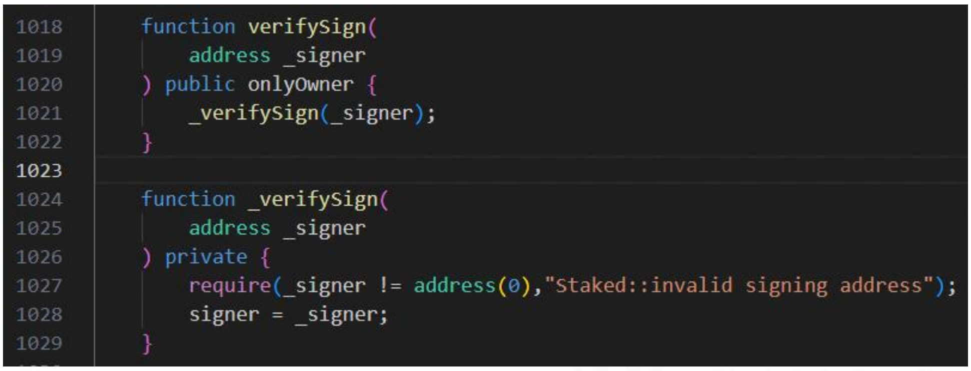
[NEWFI-5] Missing event trigger

Severity Level Info

Type Business Security

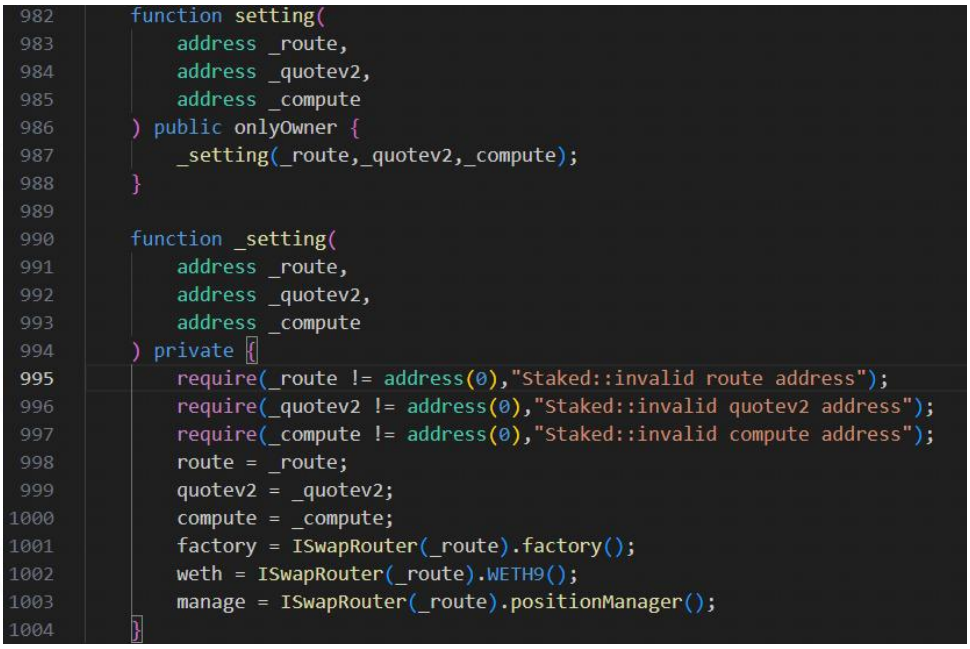
Lines StakedV3.sol#L982-1004, L1018-1029

Description Events are not triggered when important variables in the contract change, such as signer and route addresses.



```
1018    function verifySign(
1019        address _signer
1020    ) public onlyOwner {
1021        _verifySign(_signer);
1022    }
1023
1024    function _verifySign(
1025        address _signer
1026    ) private {
1027        require(_signer != address(0),"Staked::invalid signing address");
1028        signer = _signer;
1029    }
```

Figure 8 Source code of *verifySign* function (unfixed)

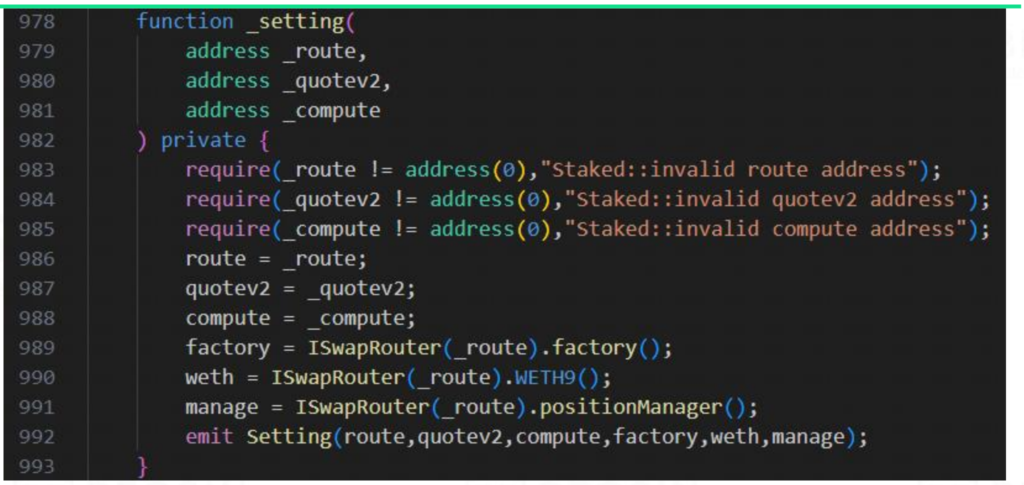


```
982     function setting(
983         address _route,
984         address _quotev2,
985         address _compute
986     ) public onlyOwner {
987         _setting(_route,_quotev2,_compute);
988     }
989
990     function _setting(
991         address _route,
992         address _quotev2,
993         address _compute
994     ) private {
995         require(_route != address(0),"Staked::invalid route address");
996         require(_quotev2 != address(0),"Staked::invalid quotev2 address");
997         require(_compute != address(0),"Staked::invalid compute address");
998         route = _route;
999         quotev2 = _quotev2;
1000        compute = _compute;
1001        factory = ISwapRouter(_route).factory();
1002        weth = ISwapRouter(_route).WETH9();
1003        manage = ISwapRouter(_route).positionManager();
1004    }
```

Figure 9 Source code of *setting* function (unfixed)

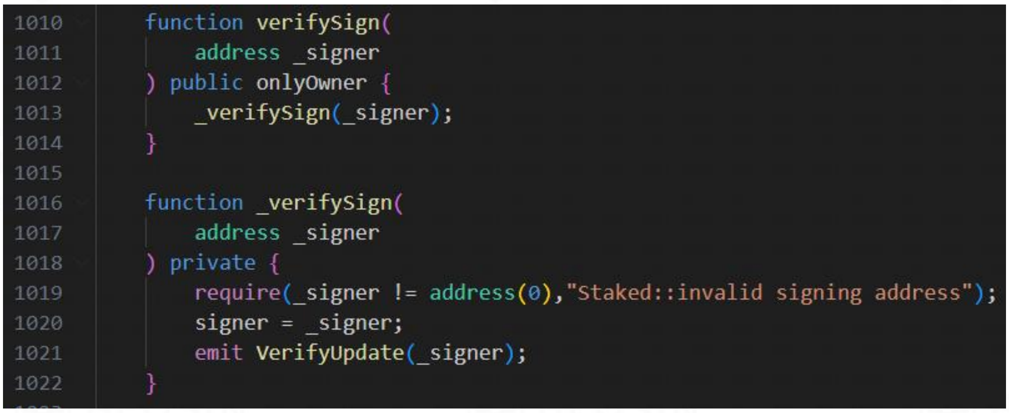
Recommendations It is recommended to add event triggering.

Status Fixed.



```
    function _setting(
        address _route,
        address _quotev2,
        address _compute
    ) private {
        require(_route != address(0),"Staked::invalid route address");
        require(_quotev2 != address(0),"Staked::invalid quotev2 address");
        require(_compute != address(0),"Staked::invalid compute address");
        route = _route;
        quotev2 = _quotev2;
        compute = _compute;
        factory = ISwapRouter(_route).factory();
        weth = ISwapRouter(_route).WETH9();
        manage = ISwapRouter(_route).positionManager();
        emit Setting(route,quotev2,compute,factory,weth,manage);
    }
```

Figure 10 Source code of *verifySign* function (fixed)



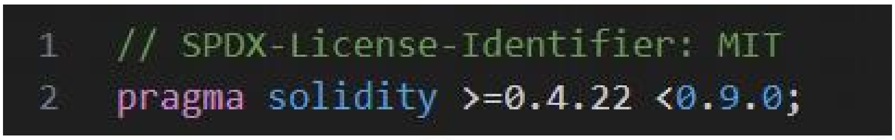
```
    function verifySign(
        address _signer
    ) public onlyOwner {
        _verifySign(_signer);
    }

    function _verifySign(
        address _signer
    ) private {
        require(_signer != address(0),"Staked::invalid signing address");
        signer = _signer;
        emit VerifyUpdate(_signer);
    }
```

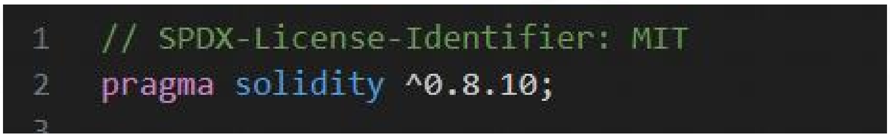
Figure 11 Source code of *setting* function (fixed)

[NEWFI-6] Compiler is not fixed

Severity Level	Info
Type	Coding Conventions
Lines	StakedV3.sol#L2

Description	The compiler version is not fixed.
	
	`// SPDX-License-Identifier: MIT`
	`pragma solidity >=0.4.22 <0.9.0;`
	Figure 12 Source code of compiler version (unfixed)

Recommendations	It is recommended to fix the compiler version.

Status	Fixed.
	
	`// SPDX-License-Identifier: MIT`
	`pragma solidity ^0.8.10;`
	Figure 13 Source code of compiler version (fixed)

3 Appendix

3.1 Vulnerability Assessment Metrics and Status in Smart Contracts

3.1.1 Metrics

In order to objectively assess the severity level of vulnerabilities in blockchain systems, this report provides detailed assessment metrics for security vulnerabilities in smart contracts with reference to CVSS 3.1 (Common Vulnerability Scoring System Ver 3.1).

According to the severity level of vulnerability, the vulnerabilities are classified into four levels: "critical", "high", "medium" and "low". It mainly relies on the degree of impact and likelihood of exploitation of the vulnerability, supplemented by other comprehensive factors to determine of the severity level.

Impact / Likelihood	Severe	High	Medium	Low
Probable	Critical	High	Medium	Low
Possible	High	High	Medium	Low
Unlikely	Medium	Medium	Low	Info
Rare	Low	Low	Info	Info

3.1.2 Degree of impact

- **Severe**

Severe impact generally refers to the vulnerability can have a serious impact on the confidentiality, integrity, availability of smart contracts or their economic model, which can cause substantial economic losses to the contract business system, large-scale data disruption, loss of authority management, failure of key functions, loss of credibility, or indirectly affect the operation of other smart contracts associated with it and cause substantial losses, as well as other severe and mostly irreversible harm.

- **High**

High impact generally refers to the vulnerability can have a relatively serious impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a greater economic loss, local functional unavailability, loss of credibility and other impact to the contract business system.

- **Medium**

Medium impact generally refers to the vulnerability can have a relatively minor impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a small amount of economic loss to the contract business system, individual business unavailability and other impact.

- **Low**

Low impact generally refers to the vulnerability can have a minor impact on the smart contract, which can pose certain security threat to the contract business system and needs to be improved.

3.1.4 Likelihood of Exploitation

- **Probable**

Probable likelihood generally means that the cost required to exploit the vulnerability is low, with no special exploitation threshold, and the vulnerability can be triggered consistently.

- **Possible**

Possible likelihood generally means that exploiting such vulnerability requires a certain cost, or there are certain conditions for exploitation, and the vulnerability is not easily and consistently triggered.

- **Unlikely**

Unlikely likelihood generally means that the vulnerability requires a high cost, or the exploitation conditions are very demanding and the vulnerability is highly difficult to trigger.

- **Rare**

Rare likelihood generally means that the vulnerability requires an extremely high cost or the conditions for exploitation are extremely difficult to achieve.

3.1.5 Fix Results Status

Status	**Description**
Fixed	The project party fully fixes a vulnerability.
Partially Fixed	The project party did not fully fix the issue, but only mitigated the issue.
Acknowledged	The project party confirms and chooses to ignore the issue.

3.2 Audit Categories

No.	Categories	Subitems
1	Coding Conventions	Redundant Code
		require/assert Usage
		Cycles Consumption
2	General Vulnerability	Integer Overflow/Underflow
		Reentrancy
		Pseudo-random Number Generator (PRNG)
		Transaction-Ordering Dependence
		DoS (Denial of Service)
		Function Call Permissions
		Returned Value Security
		Rollback Risk
		Replay Attack
		Overriding Variables
		Call Canister controllable
		Canister upgrade risk
		Third-party Protocol Interface Consistency
3	Business Security	Business Logics
		Business Implementations
		Manipulable Token Price
		Centralized Asset Control
		Asset Tradability
		Arbitrage Attack

Beosin classified the security issues of smart contracts into three categories: Coding Conventions, General Vulnerability, Business Security. Their specific definitions are as follows:

- **Coding Conventions**

Audit whether smart contracts follow recommended language security coding practices. For example, smart contracts developed in Solidity language should fix the compiler version and do not use deprecated keywords.

- **General Vulnerability**

General Vulnerability include some common vulnerabilities that may appear in smart contract projects. These vulnerabilities are mainly related to the characteristics of the smart contract itself, such as integer overflow/underflow and denial of service attacks.

- **Business Security**

Business security is mainly related to some issues related to the business realized by each project, and has a relatively strong pertinence. For example, whether the lock-up plan in the code match the white paper, or the flash loan attack caused by the incorrect setting of the price acquisition oracle.

*Note that the project may suffer stake losses due to the integrated third-party protocol. This is not something Beosin can control. Business security requires the participation of the project party. The project party and users need to stay vigilant at all times.

3.3 Disclaimer

The Audit Report issued by Beosin is related to the services agreed in the relevant service agreement. The Project Party or the Served Party (hereinafter referred to as the "Served Party") can only be used within the conditions and scope agreed in the service agreement. Other third parties shall not transmit, disclose, quote, rely on or tamper with the Audit Report issued for any purpose.

The Audit Report issued by Beosin is made solely for the code, and any description, expression or wording contained therein shall not be interpreted as affirmation or confirmation of the project, nor shall any warranty or guarantee be given as to the absolute flawlessness of the code analyzed, the code team, the business model or legal compliance.

The Audit Report issued by Beosin is only based on the code provided by the Served Party and the technology currently available to Beosin. However, due to the technical limitations of any organization, and in the event that the code provided by the Served Party is missing information, tampered with, deleted, hidden or subsequently altered, the audit report may still fail to fully enumerate all the risks.

The Audit Report issued by Beosin in no way provides investment advice on any project, nor should it be utilized as investment suggestions of any type. This report represents an extensive evaluation process designed to help our customers improve code quality while mitigating the high risks in blockchain.

3.4 About Beosin

Beosin is the first institution in the world specializing in the construction of blockchain security ecosystem. The core team members are all professors, postdocs, PhDs, and Internet elites from world-renowned academic institutions. Beosin has more than 20 years of research in formal verification technology, trusted computing, mobile security and kernel security, with overseas experience in studying and collaborating in project research at well-known universities. Through the security audit and defense deployment of more than 2,000 smart contracts, over 50 public blockchains and wallets, and nearly 100 exchanges worldwide, Beosin has accumulated rich experience in security attack and defense of the blockchain field, and has developed several security products specifically for blockchain.

BEOSIN
Blockchain Security

Official Website

https://www.beosin.com

Telegram

https://t.me/+dD8Bnqd133RmNWNl

Twitter

https://twitter.com/Beosin_com

Email

Contact@beosin.com