BEOSIN
Blockchain Security

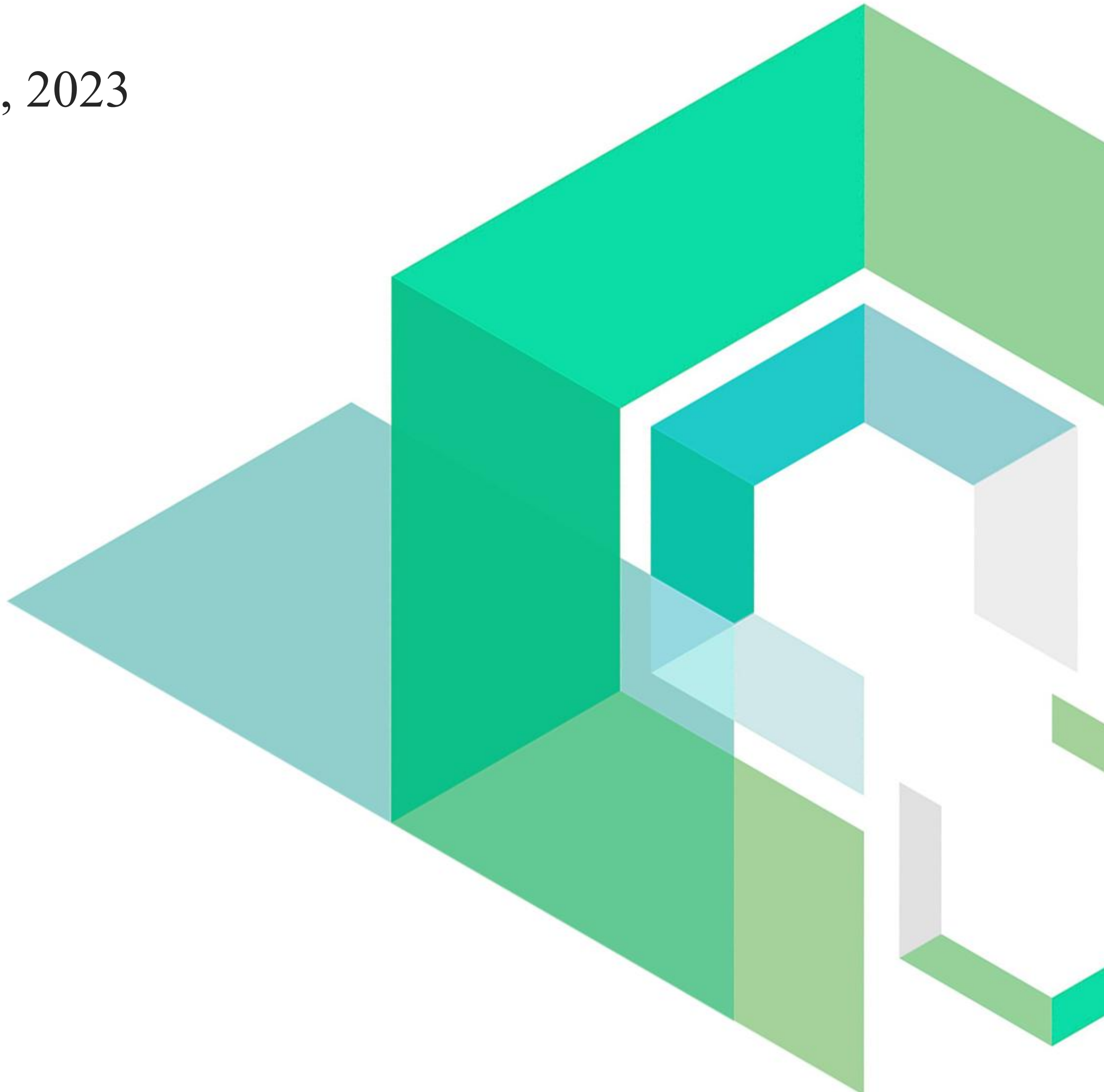
Butter

Smart Contract Security Audit

V1.0

No. 202305181022

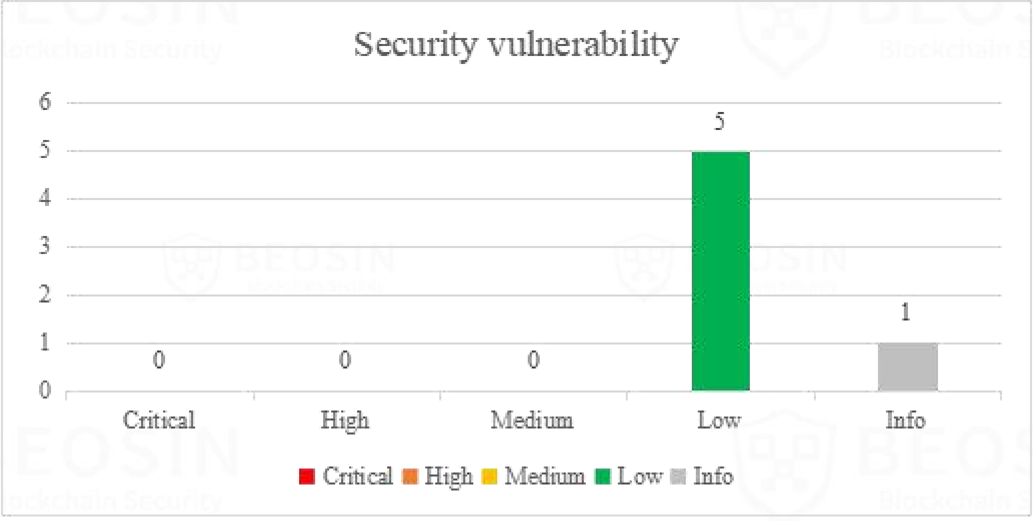
May 18th, 2023



Contents

Summary of Audit Results

After auditing, 5 Low risks and 1 Info items were identified in the Butter project. Specific audit details will be presented in the **Findings** section. Users should pay attention to the following aspects when interacting with this project:



Project Description:

1. Business overview

Butter Network supports omnichain asset swaps powered by Light-Client and ZK Technology. Butter allows omnichain asset swaps which unlocks access to dApps across chains with a single click. Butter's SDKs support all kinds of virtual assets, empowering omnichain swaps and on-chain function calls to be linked together across an ever-growing list of integrated chains and dApps. This facilitates developers and users to easily explore omnichain in no time.

1 Overview

1.1 Project Overview

Project Name	Butter
Platform	ETH, Ploygon, BSC, OP, Arbitrum, Avalanche, Fantom, Klaytn, Near
Audit Scope	https://github.com/butternetwork/butter-mos-contracts https://github.com/butternetwork/butter-router-contracts
Butter Mos Contract Commit Hash	0adb4005358e6f00f156643a6c19fe15a78df291(Initial) e2141ad3577f19eadfc3915b4fe164d1007bb46a 2bb0c4c9311434af538b33ec3ee638343f505e6d 08b6aaf0d3cf4569d1a1afd985a509df18fb6814 ec9ae9c6a8b9cdf78531744907fde61c48e811ff 4202f570144f12beaef700ea63b79cf04384cbe6 26ff543100b95048a816f634997f7e8aa8634dbb 8ae768e1ebbd09514191f9bd053ad0f41f71e687 d1ba30567712717fed9be568a76de9080eb89802 4f5ca4062fa926fb3594927534430727 42596dc5(Final)

Butter Router Contract Commit Hash	aa2fd6d3d197f700be1b64708410713906fb609e(Initial)
	aa09195dc4d48d75a42f70f02eadf3f943292b8c
	685f4dcf98bb172d24b819c499b1ebf11597d13e
	7a9c9b0b5e3d4c2d463da6d4d484c02d828a8802(Final)

1.2 Audit Overview

Audit work duration: May 4, 2023 –May 18, 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
Butter - 1	Data overflow	Low	Fixed
Butter - 2	It is not determined whether the quantity is greater than 0	Low	Fixed
Butter– 3	Does not support deflationary tokens	Low	Acknowledged
Butter– 4	Controller can burn any user's MCS token	Low	Acknowledged
Butter - 5	2FA not enabled for privileged functions	Low	Acknowledged
Butter - 6	Multiple functions missing event records	Info	Acknowledged

Status Notes:

1. Butter-3 is not fixed, might not be able to support fee tokens in the future. The project side responded that it knows the possible risks, but here it is considered that the cross-chain tokens supported by MOS are set by the project side, and support for deflationary tokens is not considered for the time being.

2. Butter-4 is not fixed and may cause the user's tokens losses if the owner changes the address of the controller from the mos contract to another address.

3. Butter-5 is not fixed and important parameters of the contract may be tampered if the function call keys of the privileged addresses are leaked.

4. Butter-6 is not fixed and may not cause any issue.

Finding Details:

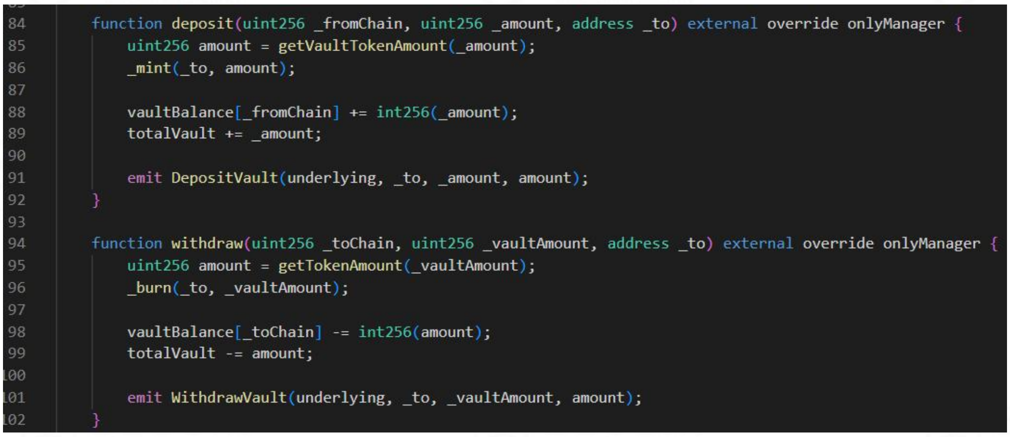
[Butter - 1] Data overflow

Severity Level Low

Type Business Security

Lines evmv2/contracts/token/VaultTokenV2.sol#L84-L102

Description When the amount is greater than the maximum value of int256, the value of vaultBalance[_fromChain] will be negative, which is inconsistent with the design

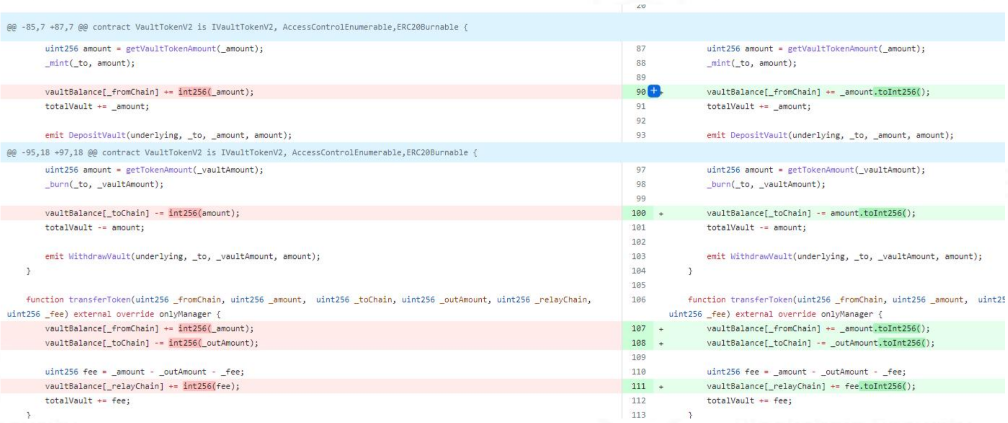


```
84      function deposit(uint256 _fromChain, uint256 _amount, address _to) external override onlyManager {
85          uint256 amount = getVaultTokenAmount(_amount);
86          _mint(_to, amount);
87
88          vaultBalance[_fromChain] += int256(_amount);
89          totalVault += _amount;
90
91          emit DepositVault(underlying, _to, _amount, amount);
92      }
93
94      function withdraw(uint256 _toChain, uint256 _vaultAmount, address _to) external override onlyManager {
95          uint256 amount = getTokenAmount(_vaultAmount);
96          _burn(_to, _vaultAmount);
97
98          vaultBalance[_toChain] -= int256(amount);
99          totalVault -= amount;
100
101         emit WithdrawVault(underlying, _to, _vaultAmount, amount);
102     }
```

Figure 1 Source code of *deposit* and *withdraw* function

Recommendations It is recommended to use a safe data conversion library, such as safecast.

Status Fixed.



```
@@ -85,7 +87,7 @@ contract VaultTokenV2 is IVaultTokenV2, AccessControlEnumerable,ERC20Burnable {
        uint256 amount = getVaultTokenAmount(_amount);
        _mint(_to, amount);

        vaultBalance[_fromChain] += int256(_amount);
        totalVault += _amount;

        emit DepositVault(underlying, _to, _amount, amount);
@@ -95,18 +97,18 @@ contract VaultTokenV2 is IVaultTokenV2, AccessControlEnumerable,ERC20Burnable {
        uint256 amount = getTokenAmount(_vaultAmount);
        _burn(_to, _vaultAmount);

        vaultBalance[_toChain] -= int256(amount);
        totalVault -= amount;

        emit WithdrawVault(underlying, _to, _vaultAmount, amount);
    }

    function transferToken(uint256 _fromChain, uint256 _amount,  uint256 _toChain, uint256 _outAmount, uint256 _relayChain,
uint256 _fee) external override onlyManager {
        vaultBalance[_fromChain] += int256(_amount);
        vaultBalance[_toChain] -= int256(_outAmount);

        uint256 fee = _amount - _outAmount - _fee;
        vaultBalance[_relayChain] += int256(fee);
        totalVault += fee;
    }
```

```
87          uint256 amount = getVaultTokenAmount(_amount);
88          _mint(_to, amount);
89
90   +      vaultBalance[_fromChain] += _amount.toInt256();
91          totalVault += _amount;
92
93          emit DepositVault(underlying, _to, _amount, amount);

97          uint256 amount = getTokenAmount(_vaultAmount);
98          _burn(_to, _vaultAmount);
99
100  +      vaultBalance[_toChain] -= amount.toInt256();
101         totalVault -= amount;
102
103         emit WithdrawVault(underlying, _to, _vaultAmount, amount);
104     }
105
106     function transferToken(uint256 _fromChain, uint256 _amount,  uint25
     uint256 _fee) external override onlyManager {
107  +      vaultBalance[_fromChain] += _amount.toInt256();
108  +      vaultBalance[_toChain] -= _outAmount.toInt256();
109
110         uint256 fee = _amount - _outAmount - _fee;
111  +      vaultBalance[_relayChain] += fee.toInt256();
112         totalVault += fee;
113     }
```

Figure 2 Source code of VaultTokenV2 contract(fixed)

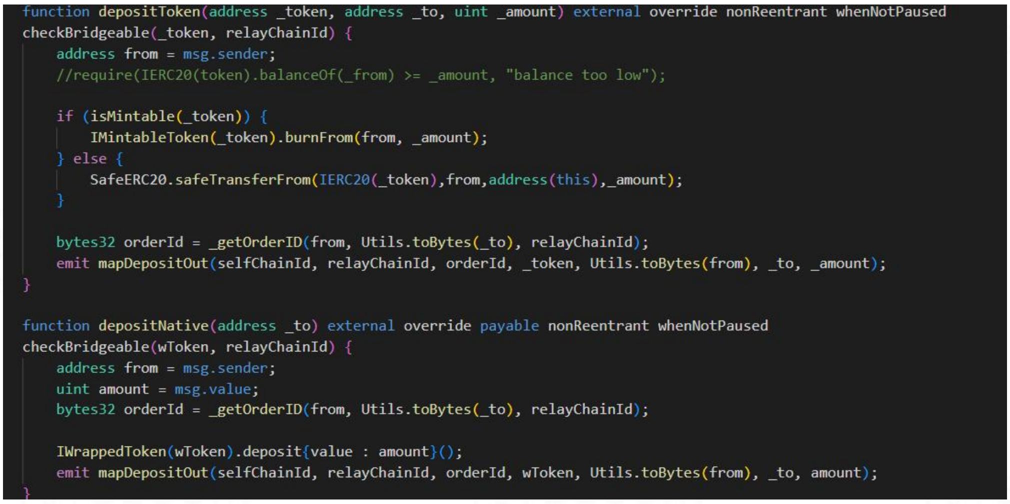
[Butter - 2] It is not determined whether the quantity is greater than 0

Severity Level Low

Type Business Security

Lines evmv2/contracts/MAPOmnichainServiceV2.sol#L152,L207,L223,

Description *swapOutToken*, *depositToken*, and *depositNative* have not judged whether the amount is greater than 0. If the amount is 0, an invalid order will be created.

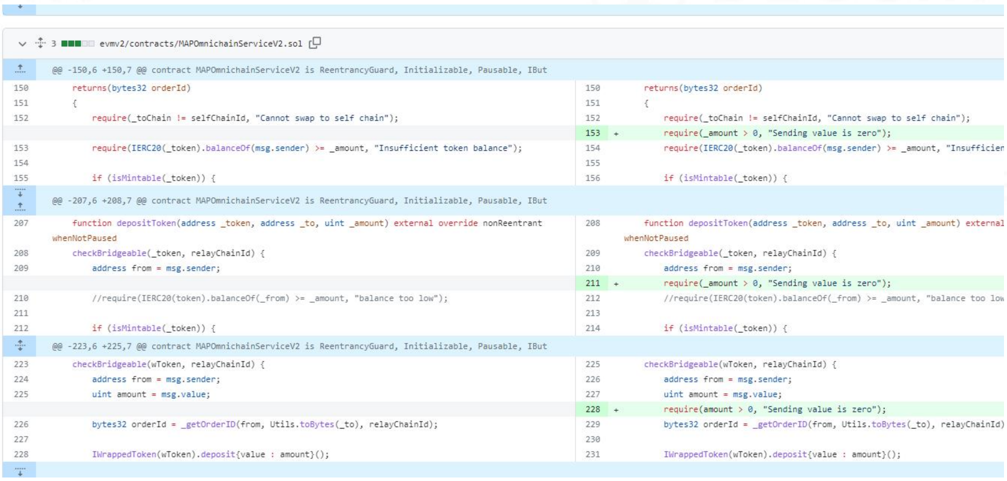


```
function depositToken(address _token, address _to, uint _amount) external override nonReentrant whenNotPaused
checkBridgeable(_token, relayChainId) {
    address from = msg.sender;
    //require(IERC20(token).balanceOf(_from) >= _amount, "balance too low");

    if (isMintable(_token)) {
        IMintableToken(_token).burnFrom(from, _amount);
    } else {
        SafeERC20.safeTransferFrom(IERC20(_token),from,address(this),_amount);
    }

    bytes32 orderId = _getOrderID(from, Utils.toBytes(_to), relayChainId);
    emit mapDepositOut(selfChainId, relayChainId, orderId, _token, Utils.toBytes(from), _to, _amount);
}

function depositNative(address _to) external override payable nonReentrant whenNotPaused
checkBridgeable(wToken, relayChainId) {
    address from = msg.sender;
    uint amount = msg.value;
    bytes32 orderId = _getOrderID(from, Utils.toBytes(_to), relayChainId);

    IWrappedToken(wToken).deposit{value : amount}();
    emit mapDepositOut(selfChainId, relayChainId, orderId, wToken, Utils.toBytes(from), _to, amount);
}
```

Figure 3 Source code of *deposit* and *depositNative* function

Recommendations It is recommended to judge that the amount must be greater than 0.

Status Fixed.



evmv2/contracts/MAPOmnichainServiceV2.sol

```
@@ -150,6 +150,7 @@ contract MAPOmnichainServiceV2 is ReentrancyGuard, Initializable, Pausable, IBut
150        returns(bytes32 orderId)
151        {
152            require(_toChain != selfChainId, "Cannot swap to self chain");
153            require(IERC20(_token).balanceOf(msg.sender) >= _amount, "Insufficient token balance");
154
155            if (isMintable(_token)) {
@@ -207,6 +208,7 @@ contract MAPOmnichainServiceV2 is ReentrancyGuard, Initializable, Pausable, IBut
207        function depositToken(address _token, address _to, uint _amount) external override nonReentrant
       whenNotPaused
208        checkBridgeable(_token, relayChainId) {
209            address from = msg.sender;
210            //require(IERC20(token).balanceOf(_from) >= _amount, "balance too low");
211
212            if (isMintable(_token)) {
@@ -223,6 +225,7 @@ contract MAPOmnichainServiceV2 is ReentrancyGuard, Initializable, Pausable, IBut
223        checkBridgeable(wToken, relayChainId) {
224            address from = msg.sender;
225            uint amount = msg.value;
226            bytes32 orderId = _getOrderID(from, Utils.toBytes(_to), relayChainId);
227
228            IWrappedToken(wToken).deposit{value : amount}();
```

```
150        returns(bytes32 orderId)
151        {
152            require(_toChain != selfChainId, "Cannot swap to self chain");
153  +         require(_amount > 0, "Sending value is zero");
154            require(IERC20(_token).balanceOf(msg.sender) >= _amount, "Insufficien
155
156            if (isMintable(_token)) {

208        function depositToken(address _token, address _to, uint _amount) external
       whenNotPaused
209        checkBridgeable(_token, relayChainId) {
210            address from = msg.sender;
211  +         require(_amount > 0, "Sending value is zero");
212            //require(IERC20(token).balanceOf(_from) >= _amount, "balance too low
213
214            if (isMintable(_token)) {

225        checkBridgeable(wToken, relayChainId) {
226            address from = msg.sender;
227            uint amount = msg.value;
228  +         require(amount > 0, "Sending value is zero");
229            bytes32 orderId = _getOrderID(from, Utils.toBytes(_to), relayChainId)
230
231            IWrappedToken(wToken).deposit{value : amount}();
```

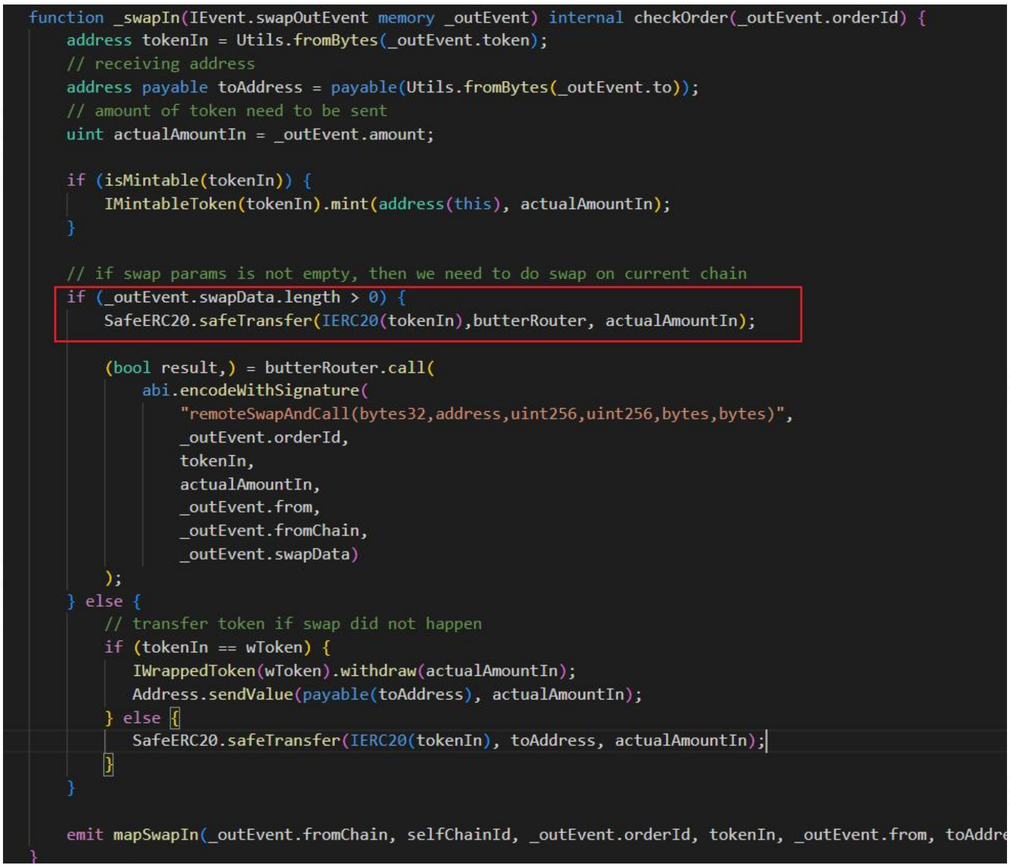
Figure 4 Source code of *deposit* and *depositNative* function(fixed)

[Butter - 3] Does not support deflationary tokens

Severity Level	Low
Type	Business Security
Lines	evmv2/contracts/MAPOmnichainServiceV2.sol#L276

Description

The contract will send actualAmountIn amount of tokens to the butterRouter contract, but if the token is a deflationary token, the actual tokens received by butterRouter will be less than actualAmountIn. This will lead to insufficient tokens for butterRouter.



```
function _swapIn(IEvent.swapOutEvent memory _outEvent) internal checkOrder(_outEvent.orderId) {
    address tokenIn = Utils.fromBytes(_outEvent.token);
    // receiving address
    address payable toAddress = payable(Utils.fromBytes(_outEvent.to));
    // amount of token need to be sent
    uint actualAmountIn = _outEvent.amount;

    if (isMintable(tokenIn)) {
        IMintableToken(tokenIn).mint(address(this), actualAmountIn);
    }

    // if swap params is not empty, then we need to do swap on current chain
    if (_outEvent.swapData.length > 0) {
        SafeERC20.safeTransfer(IERC20(tokenIn),butterRouter, actualAmountIn);

        (bool result,) = butterRouter.call(
            abi.encodeWithSignature(
                "remoteSwapAndCall(bytes32,address,uint256,uint256,bytes,bytes)",
                _outEvent.orderId,
                tokenIn,
                actualAmountIn,
                _outEvent.from,
                _outEvent.fromChain,
                _outEvent.swapData)
        );
    } else {
        // transfer token if swap did not happen
        if (tokenIn == wToken) {
            IWrappedToken(wToken).withdraw(actualAmountIn);
            Address.sendValue(payable(toAddress), actualAmountIn);
        } else {
            SafeERC20.safeTransfer(IERC20(tokenIn), toAddress, actualAmountIn);
        }
    }

    emit mapSwapIn(_outEvent.fromChain, selfChainId, _outEvent.orderId, tokenIn, _outEvent.from, toAddre
}
```

Figure 5 Source code of *_swapIn* function

Recommendations	It is recommended to compare the butterRouter balance before and after the transfer to get the actual sending amount.
Status	Acknowledged. The project side responded that it knows the possible risks, but here it is considered that the cross-chain tokens supported by MOS are set by the project side, and support for deflationary tokens is not considered for the time being.

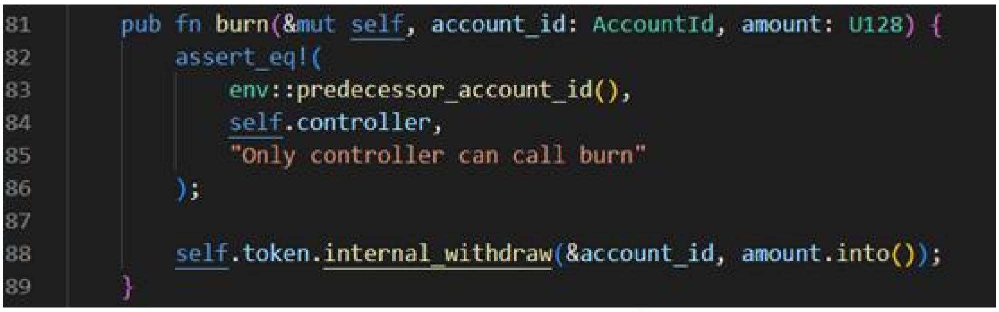
[Butter - 4] Controller can burn any user's MCS token

Severity Level Low

Type Business Security

Lines mos-token\src\lib.rs#L81-86

Description The controller authority can burn the MCS tokens of the specified user, and there is a risk of centralization.



```
81     pub fn burn(&mut self, account_id: AccountId, amount: U128) {
82         assert_eq!(
83             env::predecessor_account_id(),
84             self.controller,
85             "Only controller can call burn"
86         );
87
88         self.token.internal_withdraw(&account_id, amount.into());
89     }
```

Figure 6 Source code of *burn* function

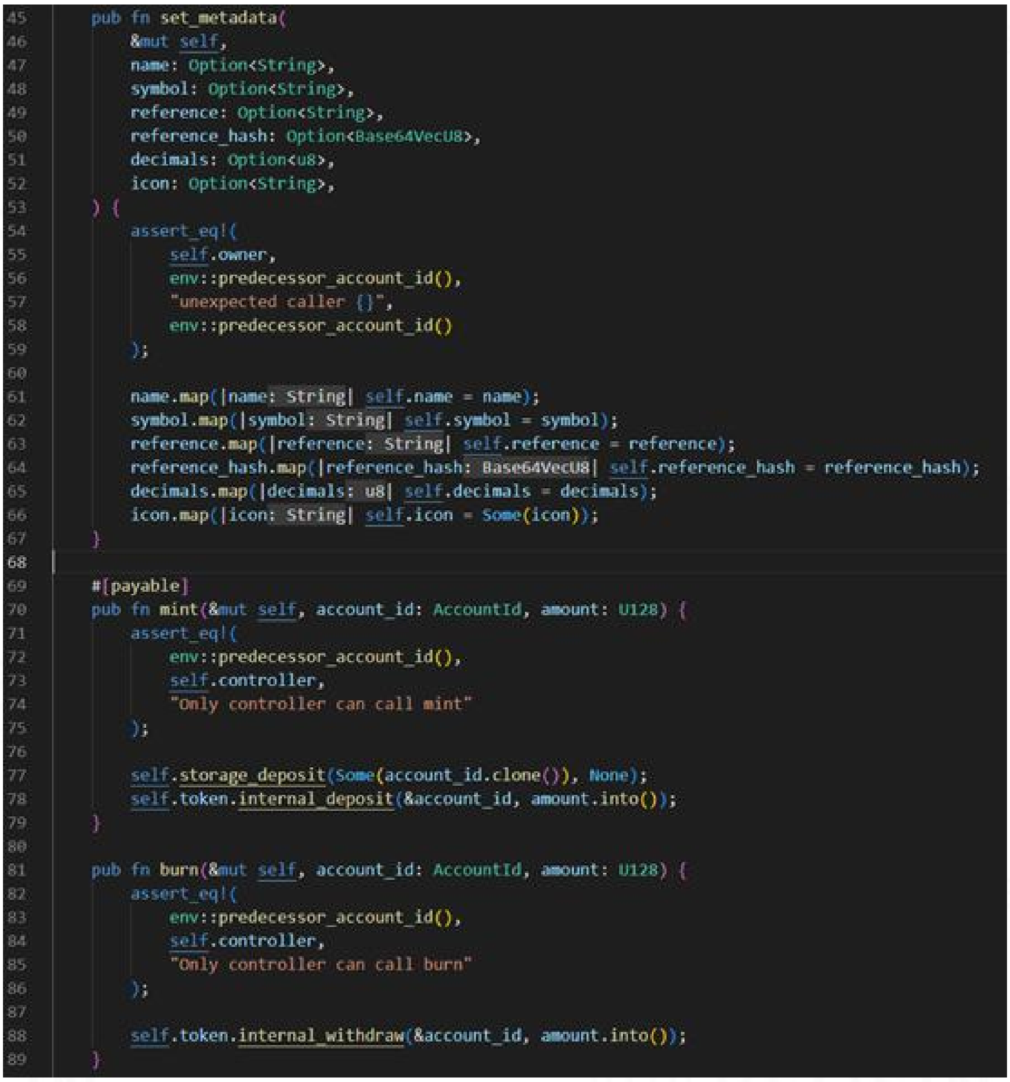
Recommendations It is recommended that each user can only *burn* the tokens owned by this address.

Status Acknowledged. The project party responded that there is a specific context calling relationship here. The controller here is a MOS contract, and the *burn* method is only allowed to be called by the MOS contract. Only when the user initiates a *transfer*/*swap* out request to the MOS contract, the MOS contract will burn the token corresponding to the user, and the MOS contract will not burn the token of the user other than the initiator.

[Butter - 5] 2FA not enabled for privileged functions

Severity Level	Low
Type	Business Security
Lines	mos-token\src\lib.rs#L45-67, 70-89, 95-117

Description Two-factor authentication is not enabled for important privileges in the contract. When the function call key is leaked, the contract may be attacked.



```
45      pub fn set_metadata(
46          &mut self,
47          name: Option<String>,
48          symbol: Option<String>,
49          reference: Option<String>,
50          reference_hash: Option<Base64VecU8>,
51          decimals: Option<u8>,
52          icon: Option<String>,
53      ) {
54          assert_eq!(
55              self.owner,
56              env::predecessor_account_id(),
57              "unexpected caller {}",
58              env::predecessor_account_id()
59          );
60
61          name.map(|name: String| self.name = name);
62          symbol.map(|symbol: String| self.symbol = symbol);
63          reference.map(|reference: String| self.reference = reference);
64          reference_hash.map(|reference_hash: Base64VecU8| self.reference_hash = reference_hash);
65          decimals.map(|decimals: u8| self.decimals = decimals);
66          icon.map(|icon: String| self.icon = Some(icon));
67      }
68
69      #[payable]
70      pub fn mint(&mut self, account_id: AccountId, amount: U128) {
71          assert_eq!(
72              env::predecessor_account_id(),
73              self.controller,
74              "Only controller can call mint"
75          );
76
77          self.storage_deposit(Some(account_id.clone()), None);
78          self.token.internal_deposit(&account_id, amount.into());
79      }
80
81      pub fn burn(&mut self, account_id: AccountId, amount: U128) {
82          assert_eq!(
83              env::predecessor_account_id(),
84              self.controller,
85              "Only controller can call burn"
86          );
87
88          self.token.internal_withdraw(&account_id, amount.into());
89      }
```

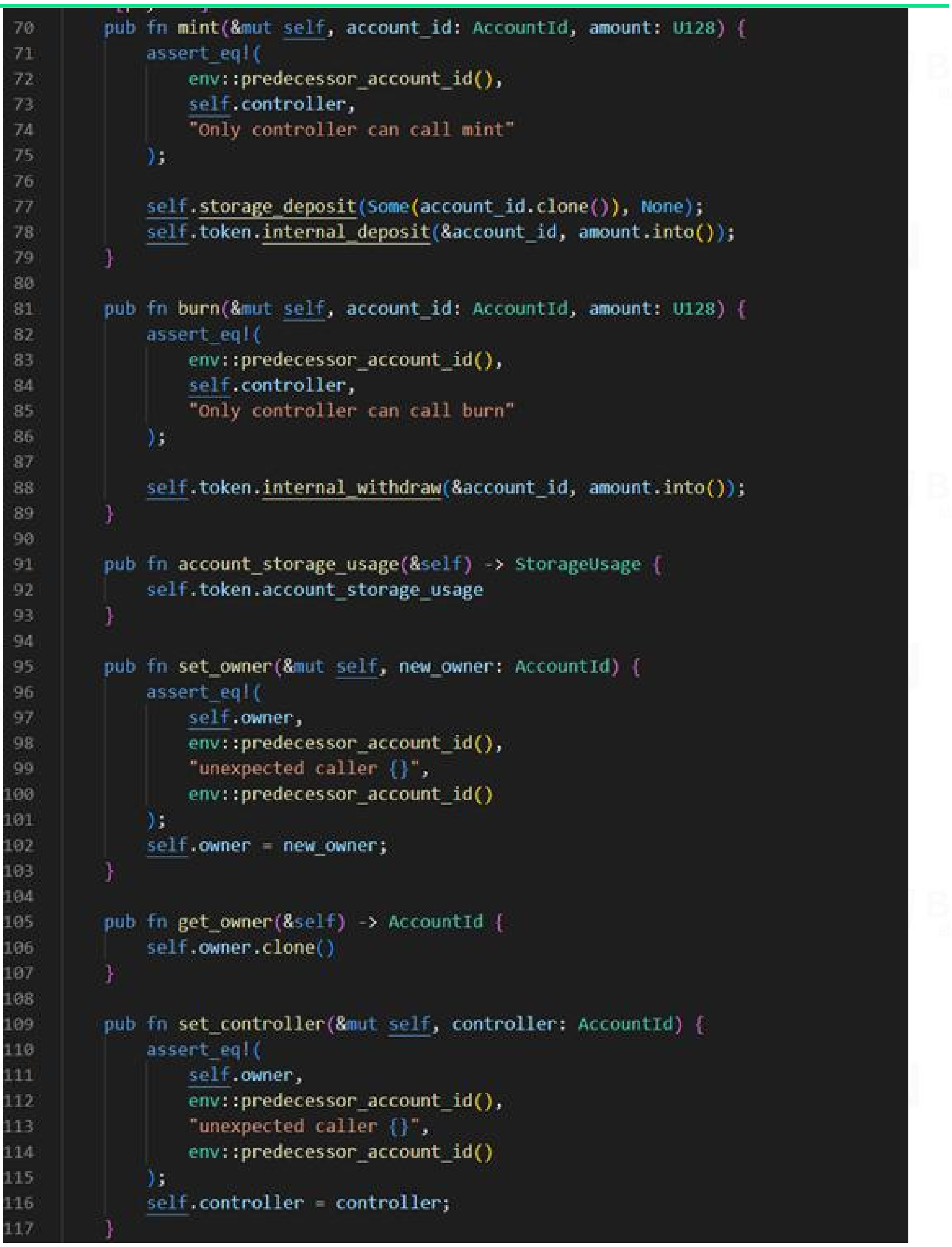
Figure 7 Source code of lib.rs

Recommendations It is recommended to add assert_one_yocto judgment in the privileged function.

Status Acknowledged. The project side responded that only the owner can call the *set_metadata* method. The owner of the MOS contract is a multi-signature contract, and the multi-signature address only supports the multi-signature of the full access key, and the leakage of the function key will not pose a threat to this method.

[Butter - 6] Multiple functions missing event records

Severity Level	Info
Type	Coding Conventions
Lines	mos-token\src\lib.rs# L70-117 map-ominichain-service\src\swap.rs#L57-155 admin-controlled\src\macros.rs#L12-15 map-ominichain-service\src\management.rs#L66-171 map-ominichain-service\src\tokens.rs#L210-243, 265-281, 320-328
Description	Missing event records when important parameters of multiple functions are changed.



```
    pub fn mint(&mut self, account_id: AccountId, amount: U128) {
        assert_eq!(
            env::predecessor_account_id(),
            self.controller,
            "Only controller can call mint"
        );

        self.storage_deposit(Some(account_id.clone()), None);
        self.token.internal_deposit(&account_id, amount.into());
    }

    pub fn burn(&mut self, account_id: AccountId, amount: U128) {
        assert_eq!(
            env::predecessor_account_id(),
            self.controller,
            "Only controller can call burn"
        );

        self.token.internal_withdraw(&account_id, amount.into());
    }

    pub fn account_storage_usage(&self) -> StorageUsage {
        self.token.account_storage_usage
    }

    pub fn set_owner(&mut self, new_owner: AccountId) {
        assert_eq!(
            self.owner,
            env::predecessor_account_id(),
            "unexpected caller {}",
            env::predecessor_account_id()
        );
        self.owner = new_owner;
    }

    pub fn get_owner(&self) -> AccountId {
        self.owner.clone()
    }

    pub fn set_controller(&mut self, controller: AccountId) {
        assert_eq!(
            self.owner,
            env::predecessor_account_id(),
            "unexpected caller {}",
            env::predecessor_account_id()
        );
        self.controller = controller;
    }
```

Figure 8 Source code of lib.rs

Recommendations	It is recommended to add corresponding events to key functions.
Status	Acknowledged. It is recommended to trigger corresponding events according to the official NEP-249 standard when important parameters are changed.

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	Compiler Version Security
		Deprecated Items
		Redundant Code
		require/assert Usage
		Gas Consumption
2	General Vulnerability	Integer Overflow/Underflow
		Reentrancy
		Pseudo-random Number Generator (PRNG)
		Transaction-Ordering Dependence
		DoS (Denial of Service)
		Function Call Permissions
		call/delegatecall Security
		Returned Value Security
		tx.origin Usage
		Replay Attack
		Overriding Variables
		Third-party Protocol Interface Consistency
3	Business Security	Business Logics
		Business Implementations
		Manipulable Token Price
		Centralized Asset Control
		Asset Tradability
		Buttertrage 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