BEOSIN
Blockchain Security

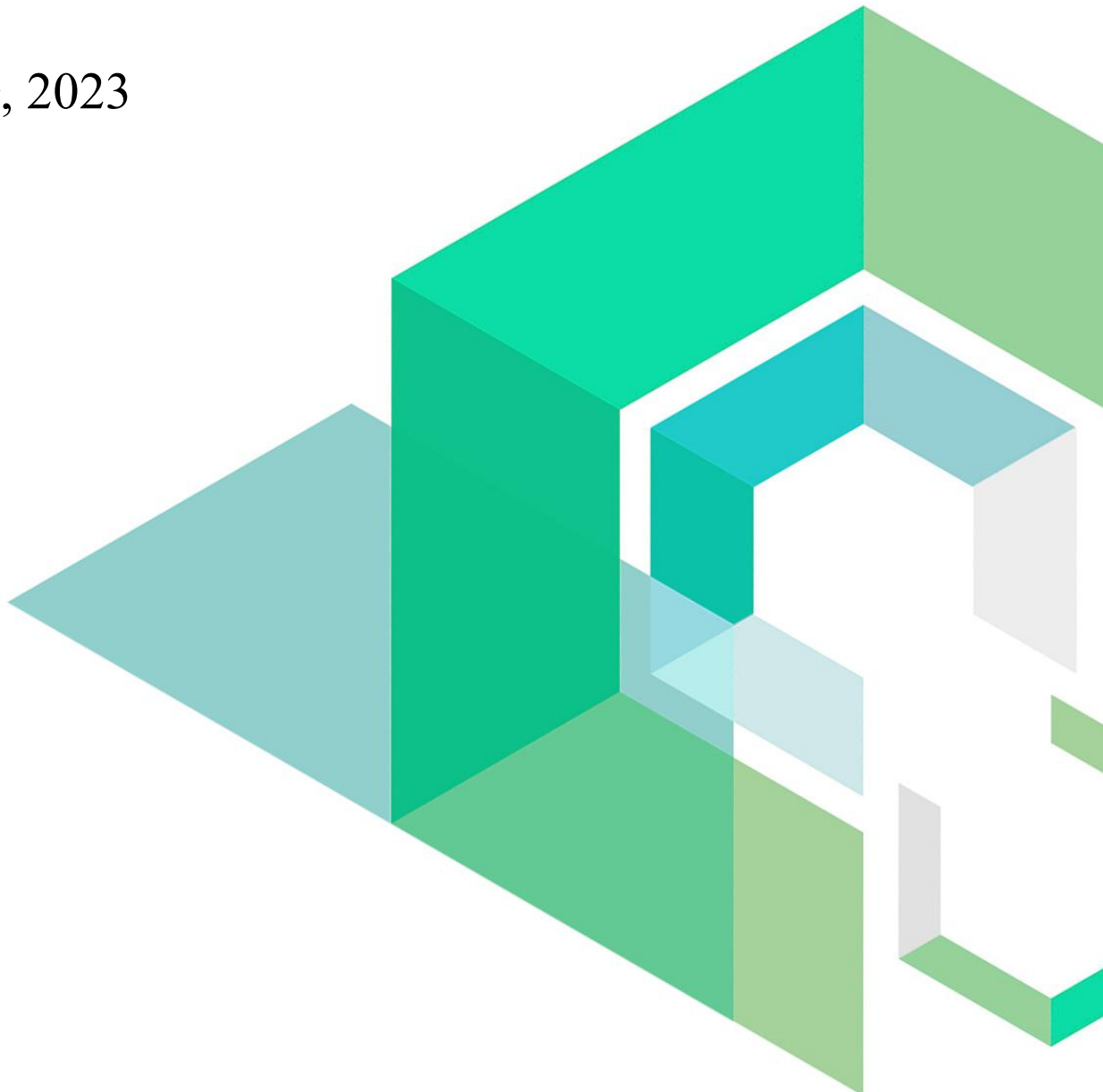
DAism

Smart Contract Security Audit

V1.0

No. 202305181349

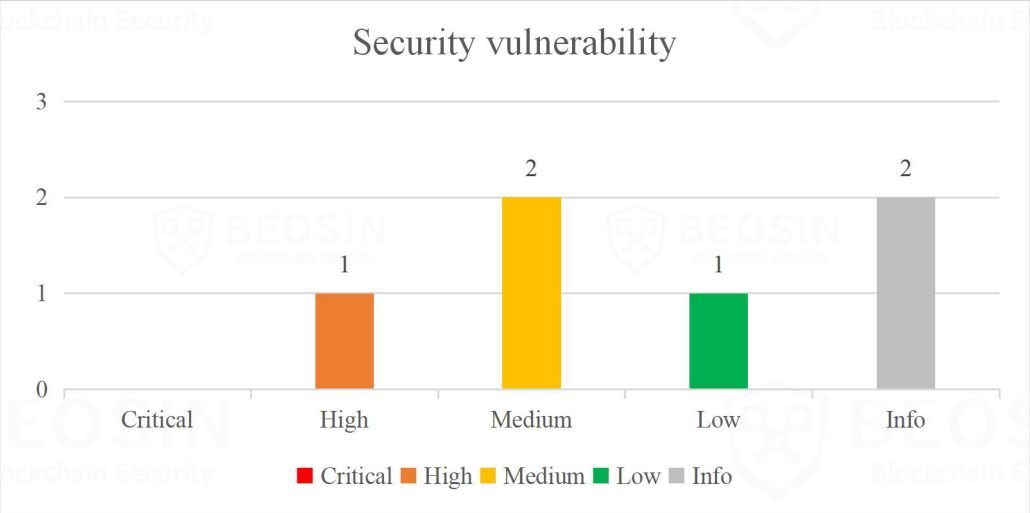
May 18th, 2023



Contents

Summary of Audit Results

After auditing, 1 High-risk and 2 Medium-risk and 1 Low-risk and 2 Info item were identified in the DAism 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. In the DAism project, Unit Token is an important token. The variable ethereumFoundation address set by the owner has arbitrary mint permissions. The official promise to set it as the Ethereum Foundation, users should pay attention.

- **Project Description:**

Business overview

DAism is a decentralized autonomous organization blockchain project that will be deployed on the ETH chain. It includes registration and creation of Dao, Dao plug-in management, and iadd network part, in which the contract will adopt the proxy upgrade mode, and the specific implementation of the contract may be different from the current audit code. On the one hand, when registering Dao, you can choose whether to issue Dao token. DaoToken will be mint to the iadd network, and the gas token part of the registration record can be exchanged for Unit token. And the number of voting accounts set during creation cannot exceed 20, and members can initiate proposals in the system and verify them with signatures. If the voting rate is 100%, it can be directly executed without plug-in checks. Adding system software in the DAism project requires owner permission, but adding plug-in software does not require permission, but modifying information requires the manager permission of the plug-in software. On the other hand, the Iadd network provides the exchange between tokens in the DAism system. The operation of the platform currency will be one-way, and it cannot be exchanged for eth after being exchanged for Unit token or Dao token. The Dao token contract will issue all Mint tokens to the iadd network, and establish a pool with the virtual Unit Token (there is no Unit token initially), and use the CPMM algorithm for calculation. The exchange in iadd will charge 5/100000 protocol fee and 195/100000 dao fee, dao fee is attributed to the corresponding dao. In the contract, the exchange of Unit token by ETH will be rewarded according to the algorithm set by the project party, and will always be exchanged at the highest price in history.

1 Overview

1.1 Project Overview

Project Name	DAism
Platform	Ethereum
Project Link	https://github.com/DAism2019/Smartcontract
Commit Hash	80d5978afbdb3ae21869abc54f992465fdf47dca 681a24473aa88fcb16932885c81309565b154153 083c1c5be7efcfb22872db2117764afa0ab1f5fa 13d5f60c563abe6b720b51e63533940bf76043e1

1.2 Audit Overview

Audit work duration: May 5, 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
DAism-1	Approve value setting error	High	Fixed
DAism-2	Miscalculation of handling fee	Medium	Fixed
DAism-3	Centralization risk	Medium	Acknowledged
DAism-4	Y variable calculation risk	Low	Fixed
DAism-5	The *_transferBatch* function missing transfer number judgment	Info	Acknowledged
DAism-6	Missing event trigger	Info	Fixed

Status Notes:

1. DAism-3 is not fixed and may cause the permission address set by the owner to be arbitrary mint Unit Token.
2. DAism-5 is not fixed and may cause a large number of transfers to fail due to the gas upper limit.

Finding Details:

[DAism-1] Approve value setting error

Severity Level	High
Type	Business Security
Lines	EIP3712.sol#L115-135
Description	In the EIP3712.sol *transferFromBatch* function, the approve value is reset to amounts, which should be set to allowance_after, which will cause approve errors and repeated transfers can be made.

```
115    function transferFromBatch(uint[] memory eip3712_ids,address[] memory senders,
116    address[] memory recipients,uint[] memory amounts)external isInUseBatch(eip3712_ids) returns(bool){
117        require(eip3712_ids.length == amounts.length && senders.length == recipients.length && senders.length == amounts.length,
118        "DAism.EIP3712._transferBatchMul: length not equal");
119        uint[] memory allowance_after = new uint[](eip3712_ids.length);
120        address[] memory spenders = new address[](eip3712_ids.length);
121        address _msg = msg.sender;
122
123        for(uint i = 0;i < eip3712_ids.length; i++){
124            spenders[i] = _msg;
125            if(allowanceGlobal[senders[i]][_msg])
126                allowance_after[i] = allowances[eip3712_ids[i]][senders[i]][_msg];
127            else{
128                uint allowance_i = allowances[eip3712_ids[i]][senders[i]][_msg];
129                require(allowance_i >= amounts[i],"DAism.EIP3712._transferBatchMul: insufficient allowance");
130                allowance_after[i] = allowance_i - amounts[i];
131            }
132
133        }
134        _transferBatch(eip3712_ids, senders, recipients, amounts);
135        _approveBatch(eip3712_ids, senders, spenders, amounts);
136        return true;
137    }
```

Figure 1 Source code of *transferFromBatch* function (unfixed)

Recommendations	It is recommended to modify the variable to the correct approve value.
Status	Fixed.

```
115    function transferFromBatch(uint[] memory eip3712_ids,address[] memory senders,
116    address[] memory recipients,uint[] memory amounts)external isInUseBatch(eip3712_ids) returns(bool){
117        require(eip3712_ids.length == amounts.length && senders.length == recipients.length && senders.length == amounts.length,
118        "DAism.EIP3712._transferBatchMul: length not equal");
119        uint[] memory allowance_after = new uint[](eip3712_ids.length);
120        address[] memory spenders = new address[](eip3712_ids.length);
121        address _msg = msg.sender;
122
123        for(uint i = 0;i < eip3712_ids.length; i++){
124            spenders[i] = _msg;
125            if(allowanceGlobal[senders[i]][_msg])
126                allowance_after[i] = allowances[eip3712_ids[i]][senders[i]][_msg];
127            else{
128                uint allowance_i = allowances[eip3712_ids[i]][senders[i]][_msg];
129                require(allowance_i >= amounts[i],"DAism.EIP3712._transferBatchMul: insufficient allowance");
130                allowance_after[i] = allowance_i - amounts[i];
131            }
132
133        }
134        _transferBatch(eip3712_ids, senders, recipients, amounts);
135
136        _approveBatch(eip3712_ids, senders, spenders, allowance_after);
137
138        return true;
139    }
```

Figure 2 Source code of *transferFromBatch* function (fixed)

[DAism-2] Miscalculation of handling fee

Severity Level	Medium
Type	Business Security
Lines	UnitToken.sol#L165-182
Description	The *daoTokenToUnitToken* function will not deduct the handling fee when the user exchanges tokens.

```
165     function daoTokenToUnitToken(uint min_amount,uint eip3712_token_amount,
166     uint pool_id, address recipient) public onlyPoolOpen(pool_id) returns(uint) {
167
168         Pool memory pool = pools[pool_id];
169         IEIP3712(daoToken).transferFrom(pool_id,msg.sender,address(this),eip3712_token_amount);
170         eip3712_token_amount = _feeToDao(eip3712_token_amount,pool_id);
171
172         uint output_amount = uint(pool.eip3712_supply).getOutput(pool.unit_token_supply,eip3712_token_amount);
173         _changePool(0, output_amount, eip3712_token_amount, 0, pool_id);
174         output_amount = _feeToProtocol(output_amount);
175
176         require(output_amount >= min_amount,"DAism._IADD.daoTokenToUnitToken: under minamount");
177         IERC20(unitToken).transfer(recipient,output_amount);
178
179         emit DaoTokenToUnitToken(msg.sender,pool_id,recipient,eip3712_token_amount,output_amount);
180
181         return output_amount;
182     }
```

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

Recommendations	It is recommended to update the output_amount before transfer.
Status	Fixed.

```
172     function daoTokenToUnitToken(uint min_amount,uint eip3712_token_amount,
173     uint pool_id, address recipient) public onlyPoolOpen(pool_id) returns(uint) {
174
175         Pool memory pool = pools[pool_id];
176         IEIP3712(daoToken).transferFrom(pool_id,msg.sender,address(this),eip3712_token_amount);
177
178         uint output_amount = uint(pool.eip3712_supply).getOutput(pool.unit_token_supply,eip3712_token_amount);
179         _changePool(0, output_amount, eip3712_token_amount, 0, pool_id);
180
181         _fee(output_amount - output_amount * (100000 - 200) / 100000,pool_id);
182         uint can_get_amount = output_amount * (100000 - 200) / 100000;
183
184         require(can_get_amount >= min_amount,"DAism._IADD.daoTokenToUnitToken: under minamount");
185         IERC20(unitToken).transfer(recipient,can_get_amount);
186
187         emit DaoTokenToUnitToken(msg.sender,pool_id,recipient,eip3712_token_amount,can_get_amount);
188
189         return output_amount;
190     }
```

Figure 4 Source code of *daoTokenToUnitToken* function (fixed)

[DAism-3] Centralization risk

Severity Level	Medium
Type	Business Security
Lines	UnitToken.sol#L92-103
Description	The ethereumFoundation permissions in the contract can mint Unit tokens arbitrarily, and the centralized permissions is too large.

```
92    function controlEthereumFounddation(address ethereum_foundation) external onlyOwner {
93        ethereumFoundation = ethereum_foundation;
94    }
95
96    /**
97    requirements:
98    '-' only EthereumFounddation. */
99    function mint(address account, uint amount) external {
100       require(msg.sender == ethereumFoundation,"DAism.UnitToken.mint: only ethereumFoundati
101       _mint(account,amount);
102   }
103
```

Figure 5 Source code of *controlEthereumFounddation* function

Recommendations	It is recommended to use a multi-sig wallet to manage ethereumFoundation addresses.
Status	Acknowledged. The official reply will be set to Ethereum Foundation, but the owner can modify this address.

[DAism-4] Y variable calculation risk

Severity Level	Low
Type	Business Security
Lines	UnitTokenMath.sol#L23-37
Description	In the unitToken contract, the *swap* function calculation will continue to reduce the y value, and the minimum value will reach zero. However, if the Y value reaches zero during the calculation, the calculation will fail, so that the contract function cannot be used.

```
123     function swap(address to) external payable returns(uint) {
124         uint eth_amount_input = msg.value;
125         (uint amount,uint highest_price_in_history,uint extra_reward) = getOutputAmount(eth_amount_input);
126         x += eth_amount_input;
127         y -= extra_reward;
128 
129         burnAmount[msg.sender] += eth_amount_input;
130         burnReward[msg.sender] += amount - extra_reward;
131 
132         updateHighestPrice(highest_price_in_history);
133         _mint(to,amount);
134         emit Swap(to,eth_amount_input,amount);
135         return amount;
136     }
```

Figure 6 Source code of *swap* function

```
23  function commulateETHInputAmount(uint x, uint y, uint highestPriceInHistory, uint b) internal pure returns(uint a,uint b1) {
24      uint y_b_highestPriceInHistoryX = y + b + highestPriceInHistory * x;
25      uint two_b1 = y_b_highestPriceInHistoryX - sqrt(y_b_highestPriceInHistoryX * y_b_highestPriceInHistoryX - 4 * b * y);
26      b1 = two_b1 / 2;
27      a = (x * b1) / (y - b1);
28 
29      //check a
30      uint b_check = highestPriceInHistory * a + (y * a) / (x + a);
31      if(b_check + (y * (a+1))/(x+1+a) - b1 + highestPriceInHistory < b){
32          a ++;
33          b1 = (y * (a+1))/(x+1+a);
34      }
35 
36      if(b_check + (y*(a+100))/(x+a+100) + 100 * highestPriceInHistory < b)
37          revert("deviation too large");
38  }
```

Figure 7 Source code of *commulateETHInputAmount* function (unfixed)

Recommendations	It is recommended to increase the judgment of 0 value
Status	Fixed.

```
23  function commulateETHInputAmount(uint x, uint y, uint highestPriceInHistory, uint b) internal pure returns(uint a,uint b1) {
24 
25      if(y == 0)
26          return (b / highestPriceInHistory, 0);
27 
28      uint y_b_highestPriceInHistoryX = y + b + highestPriceInHistory * x;
29      uint two_b1 = y_b_highestPriceInHistoryX - sqrt(y_b_highestPriceInHistoryX * y_b_highestPriceInHistoryX - 4 * b * y);
30      b1 = two_b1 / 2;
31      a = (x * b1) / (y - b1);
32 
33      //check a
34      uint b_check = highestPriceInHistory * a + (y * a) / (x + a);
35      if(b_check + (y * (a+1))/(x+1+a) - b1 + highestPriceInHistory < b){
36          a ++;
37          b1 = (y * (a+1))/(x+1+a);
38      }
39 
40      if(b_check + (y*(a+100))/(x+a+100) + 100 * highestPriceInHistory < b)
41          revert("deviation too large");
42  }
```

Figure 8 Source code of *commulateETHInputAmount* function (fixed)

[DAism-5] The *_transferBatch* function missing transfer number judgment

Severity Level	Info
Type	Business Security
Lines	EIP3712.sol#L167-182
Description	In the *_transferBatch* function of the EIP3712 contract, the number of transfer users is not judged. Due to the block gas limit, if the transfer amount is too large, the transaction will fail.

```
164     function _transferBatch(uint[] memory eip3712_ids,address[] memory senders,
165     address[] memory recipients,uint[] memory amounts) internal virtual{
166
167         for(uint i = 0;i < eip3712_ids.length;i++){
168             require(senders[i] != address(0),"DAism.EIP3712._transferBatch:sender is a zero address");
169             require(recipients[i] != address(0),"DAism.EIP3712._transferBatch:recipient is a zero address");
170             if(amounts[i] == 0 || senders[i] == recipients[i])
171                 continue;
172             uint balance_i = balances[eip3712_ids[i]][senders[i]];
173             require(balance_i >= amounts[i],"DAism.EIP3712._transferBatch: Insufficient balance");
174             balances[eip3712_ids[i]][senders[i]] = balance_i - amounts[i];
175             balances[eip3712_ids[i]][recipients[i]] = balances[eip3712_ids[i]][recipients[i]] + amounts[i];
176         }
177         emit TransferBatch(eip3712_ids,senders,recipients,amounts);
178     }
```

Figure 9 Source code of *_transferBatch* function

Recommendations	It is recommended to increase the judgment of eip3712.length.
Status	Acknowledged.

[DAism-6] Missing event trigger

Severity Level	Info
Type	Business Security
Lines	UnitToken.sol#L67-69
Description	The *controlEthUsdc* function will change important address variables related to Uniswap V3, and missing event trigger records.

```
67    function controlEthUsdc(address eth_usdc) external onlyOwner {
68        ethUsdc = eth_usdc;
69    }
70
```

Figure 10 Source code of *controlEthUsdc* function (unfixed)

Recommendations	It is recommended to add event triggering.
Status	Fixed.

```
66    function init_eth_usdcs(address[] memory _eth_usdcs) external onlyOwner {
67        require(eth_usdcs.length == 0 && _eth_usdcs.length > 0);
68        eth_usdcs = _eth_usdcs;
69    }
70
```

Figure 11 Source code of *init_eth_usdcs* function

```
80    function controlEthUsdc(uint index) external onlyOwner {
81        if(index == 0)
82            ethUsdc = address(0);
83        else
84            ethUsdc = eth_usdcs[index];
85        emit ControlEthUsdc(ethUsdc);
86
87    }
```

Figure 12 Source code of *controlEthUsdc* function (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.

Likelihood \ Impact	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