Xunion Swap

Smart Contract Security Audit

No. 202405211405

May 21th, 2024

SECURING BLOCKCHAIN ECOSYSTEM

WWW.BEOSIN.COM

Contents

1 Overview **4**

1.1 Project Overview 4

1.2 Audit Overview 4

1.3 Audit Method 4

2 Findings **6**

[Xunion Swap-01] Reserve Update Vulnerability 7

[Xunion Swap-02] Denial of Service 10

[Xunion Swap-03] Parameter acquisition exception 12

[Xunion Swap-04] Price updates are not synchronized 14

[Xunion Swap-05] Incorrect use of variables 15

[Xunion Swap-06] Swap efficiency is too low 17

[Xunion Swap-07] No function to handle ETH transfers 18

[Xunion Swap-08] Centralization risk 19

[Xunion Swap-09] Redundant Code 20

3 Appendix **21**

3.1 Vulnerability Assessment Metrics and Status in Smart Contracts 21

3.2 Audit Categories 24

3.3 Disclaimer 26

3.4 About Beosin 27

Summary of Audit Results

After auditing, 1 Critical-risk,3 High-risk,1 Medium-risk,1 Low-risk and 3 Info were identified in the Xunion Swap project. Specific audit details will be presented in the Findings section. Users should pay attention to the following aspects when interacting with this project:

1 Overview

1.1 Project Overview

Project Name	Xunion Swap
Project Language	Solidity
File Hash (Sha256)	xunionswapcore.sol: 461a8d270176879da3b8fec23ab759b78747da779834ba16e34756b25111ee32 adf6086ad680043573463e50bacf59da64dc7166c79c289e9c3462c8f6bf8e19 xunionswapfactory.sol: 7d3a58b34761c7ad22d6614d0f761b93abb7d2c1d41ad4a324f2ba2c7f5ab1b6 25f7870968170b0546947b2838de70bc17e553accb8bd073255cab4a8b04a9f3 xunionswaplpmanagement.sol: 843dd75135599edc48f80efddb9493e8d7f2de76b9faaaec1f75226ef24970ee 75ac9895ae42eb3bbb1c0efd5890dbdce657aea0eaf0ee87d6853adc0a2e6f42 xunionswaplpvaults.sol: a44681e513b3b5ee7f8775f06397890f30ba5839c4b611fac02d4d0c06f1c3ba c8b5a1ecd4c89cc178e05391f57df5f7298ef67d3617ca5718833ecc884d7a3b xunionswappair.sol: 13571f3dd6cf000859b56ab497a52e2aab7e975af0df701f10784d8eac7bedc3 xunionswapvaults.sol: 1fbaaf502d7622215eb45464220daffe3b82c95d3c3f0dca75d69c8be75758ff a61469d0f5a134bc9c19249fd542bcc3892d192c4c9ac45f2de043dd08f93889

1.2 Audit Overview

Audit work duration: May 7, 2024 – May 21, 2024

Audit team: Beosin Security Team

1.3 Audit Method

The audit methods are as follows:

1. Formal Verification

Formal verification is a technique that uses property-based approaches for testing and verification. Property specifications define a set of rules using Beosin's library of security expert rules. These rules call into the contracts under analysis and make various assertions about their behavior. The rules of the specification play a crucial role in the analysis. If the rule is violated, a concrete test case is provided to demonstrate the violation.

2. Manual Review

Using manual auditing methods, the code is read line by line to identify potential security issues. This ensures that the contract's execution logic aligns with the client's specifications and intentions, thereby safeguarding the accuracy of the contract's business logic.

The manual audit is divided into three groups to cover the entire auditing process:

The Basic Testing Group is primarily responsible for interpreting the project's code and conducting comprehensive functional testing.

The Simulated Attack Group is responsible for analyzing the audited project based on the collected historical audit vulnerability database and security incident attack models. They identify potential attack vectors and collaborate with the Basic Testing Group to conduct simulated attack tests.

The Expert Analysis Group is responsible for analyzing the overall project design, interactions with third parties, and security risks in the on-chain operational environment. They also conduct a review of the entire audit findings.

3. Static Analysis

Static analysis is a method of examining code during compilation or static analysis to detect issues. Beosin-VaaS can detect more than 100 common smart contract vulnerabilities through static analysis, such as reentrancy and block parameter dependency. It allows early and efficient discovery of problems to improve code quality and security.

2 Findings

Index	Risk description	Severity level	Status
Xunion Swap-01	Reserve Update Vulnerability	Critical	Fixed
Xunion Swap-02	Denial of Service	High	Fixed
Xunion Swap-03	Parameter acquisition exception	High	Fixed
Xunion Swap-04	Price updates are not synchronized	High	Fixed
Xunion Swap-05	Incorrect use of variables	Medium	Fixed
Xunion Swap-06	Swap efficiency is too low	Low	Fixed
Xunion Swap-07	No function to handle ETH transfers	Info	Fixed
Xunion Swap-08	Centralization risk	Info	Fixed
Xunion Swap-09	Redundant Code	Info	Fixed

Finding Details:

[Xunion Swap-01] Reserve Update Vulnerability

Severity Level	**Critical**
Type	Business Security
Lines	xunionswapvaults.sol #L215-L282
Description	In the `exchange` function,when exchanging through multiple pairs, only the initial token `reserves` and the final token `reserves` (inputtoken and outputtoken) are updated, without updating the token `reserves` during the intermediate swap process. Attackers can create pairs using their own created tokens, and use this token to exchange other valuable tokens through multiple pairs. After exchanging, since the intermediate `reserves` are not updated, attackers can directly extract all the added tokens.

```
function exchange(structlibrary.exVaults memory _exVaults,uint
deadline) public lock returns(uint){
        ......
         for(i=0;i<_exVaults.tokens.length-1;i++){
             ......
             if(i==0){
                 if(getLpInputTokenSlot(_lp[i],_exVaults.tokens[i])){
                     totalTokenInVaults[0] -= inputAmount[0];
                     reserves[_lp[i]].reserve[0] += inputAmount[0] *
relativeTokenUpperLimit[reserveAddr[0]] / totalTokenInVaults[0];
                     relativeTokenUpperLimit[reserveAddr[0]] +=
inputAmount[0] * relativeTokenUpperLimit[reserveAddr[0]] /
totalTokenInVaults[0];
                 }else{
                     totalTokenInVaults[1] -= inputAmount[1];
                     reserves[_lp[i]].reserve[1] += inputAmount[0] *
relativeTokenUpperLimit[reserveAddr[1]] / totalTokenInVaults[1];
                     relativeTokenUpperLimit[reserveAddr[1]] +=
inputAmount[0] * relativeTokenUpperLimit[reserveAddr[1]] /
totalTokenInVaults[1];
                 }
             }
             if(i==_exVaults.tokens.length-2){
```

```
                if(getLpInputTokenSlot(_lp[i],_exVaults.tokens[i])){
                    reserves[_lp[i]].reserve[1] -=
outputAmount[_exVaults.tokens.length-2] *
relativeTokenUpperLimit[reserveAddr[1]] / totalTokenInVaults[1];
                    relativeTokenUpperLimit[reserveAddr[1]] -=
outputAmount[_exVaults.tokens.length-2] *
relativeTokenUpperLimit[reserveAddr[1]] / totalTokenInVaults[1];
                }else{
                    reserves[_lp[i]].reserve[0] -=
outputAmount[_exVaults.tokens.length-2] *
relativeTokenUpperLimit[reserveAddr[0]] / totalTokenInVaults[0];
                    relativeTokenUpperLimit[reserveAddr[0]] -=
outputAmount[_exVaults.tokens.length-2] *
relativeTokenUpperLimit[reserveAddr[0]] / totalTokenInVaults[0];
                }
            }
......
}
```

Recommendation	During the swap process, update the reserves of all tokens involved in the path.
Status	**Fixed.**

```
function exchange(structlibrary.exVaults memory _exVaults,uint
deadline) public lock returns(uint){
       ......
        for(i=0;i<_exVaults.tokens.length-1;i++){
            ......
            if(getLpInputTokenSlot(_lp[i],_exVaults.tokens[i])){
                if(i==0){
                    totalTokenInVaults[0] -= inputAmount[i];
                }
                reserves[_lp[i]].reserve[0] += inputAmount[i] *
relativeTokenUpperLimit[reserveAddr[0]] / totalTokenInVaults[0];
                relativeTokenUpperLimit[reserveAddr[0]] +=
inputAmount[i] * relativeTokenUpperLimit[reserveAddr[0]] /
totalTokenInVaults[0];
                reserves[_lp[i]].reserve[1] -= outputAmount[i] *
relativeTokenUpperLimit[reserveAddr[1]] / totalTokenInVaults[1];
                relativeTokenUpperLimit[reserveAddr[1]] -=
outputAmount[i] * relativeTokenUpperLimit[reserveAddr[1]] /
```

```
totalTokenInVaults[1];
            }else{
                if(i==0){
                    totalTokenInVaults[1] -= inputAmount[i];
                }
                reserves[_lp[i]].reserve[1] += inputAmount[i] *
relativeTokenUpperLimit[reserveAddr[1]] / totalTokenInVaults[1];
                relativeTokenUpperLimit[reserveAddr[1]] +=
inputAmount[i] * relativeTokenUpperLimit[reserveAddr[1]] /
totalTokenInVaults[1];
                reserves[_lp[i]].reserve[0] -= outputAmount[i] *
relativeTokenUpperLimit[reserveAddr[0]] / totalTokenInVaults[0];
                relativeTokenUpperLimit[reserveAddr[0]] -=
outputAmount[i] * relativeTokenUpperLimit[reserveAddr[0]] /
totalTokenInVaults[0];
            }
            emit XUnionExchange(_exVaults.tokens[i],
_exVaults.tokens[i+1], inputAmount[i], outputAmount[i]);
        }
        ......
    }
```

[Xunion Swap-02] Denial of Service

Severity Level	High
Type	General Vulnerability
Lines	xunionswapvaults.sol #L94-L132
Description	In the incrementLpAmount function, initializes reserve and relateTokenUpperLimit under the condition that totalTokenInVaults=0. Anyone can directly transfer tokens to the contract before initializing the reserve, which will result in the reserve and relativeTokenUpperLimit never being initialized, making the corresponding tokens unable to increase liquidity and core business unusable.
Recommendation	When initializing, avoid using the contract token balance to determine whether

```
    function increaseLpAmount(address _lp,uint[2] memory
_reserveIn,uint _lpAdd) external onlyLpManager{
        require(reserves[_lp].assetAddr[0] != address(0),"X Swap
Vaults: Cant be Zero Tokens");
        address[2] memory reserveAddr = getLpPair( _lp) ;
        uint[2] memory totalTokenInVaults;
        totalTokenInVaults[0] =
IERC20(reserveAddr[0]).balanceOf(address(this));
        totalTokenInVaults[1] =
IERC20(reserveAddr[1]).balanceOf(address(this));
        if(reserves[_lp].reserve[0]==0&&reserves[_lp].reserve[1]==0){
            if(totalTokenInVaults[0] == 0){
                reserves[_lp].reserve[0] = 1 ether;
                relativeTokenUpperLimit[reserveAddr[0]] = 1 ether;
            }else{
                require(totalTokenInVaults[0]>0, "X Swap
Vaults:totalTokenInVaults need >0");
                reserves[_lp].reserve[0] += _reserveIn[0] *
relativeTokenUpperLimit[reserveAddr[0]] / totalTokenInVaults[0];
                relativeTokenUpperLimit[reserveAddr[0]] +=
_reserveIn[0] * relativeTokenUpperLimit[reserveAddr[0]]/
totalTokenInVaults[0];
            }
......
}
```

	it has been initialized. Please reset the conditions, such as adding variables to determine whether it has been initialized.
Status	**Fixed.**

```
function increaseLpAmount(address _lp,uint[2] memory _reserveIn,uint
_lpAdd) external onlyLpManager{
        require(reserves[_lp].assetAddr[0] != address(0),"X Swap
Vaults: Cant be Zero Tokens");
        address[2] memory reserveAddr = getLpPair( _lp) ;
        uint[2] memory totalTokenInVaults;
        totalTokenInVaults[0] =
IERC20(reserveAddr[0]).balanceOf(address(this)) - _reserveIn[0];
        totalTokenInVaults[1] =
IERC20(reserveAddr[1]).balanceOf(address(this)) - _reserveIn[1];
        if(reserves[_lp].reserve[0]==0&&reserves[_lp].reserve[1]==0){
            if(relativeTokenUpperLimit[reserveAddr[0]] == 0){
                reserves[_lp].reserve[0] = 1 ether;
                relativeTokenUpperLimit[reserveAddr[0]] = 1 ether;
            }else {
                // require(totalTokenInVaults[0]>0, "X Swap
Vaults:totalTokenInVaults need >0");
                reserves[_lp].reserve[0] += _reserveIn[0] *
relativeTokenUpperLimit[reserveAddr[0]] / totalTokenInVaults[0];
                relativeTokenUpperLimit[reserveAddr[0]] +=
_reserveIn[0] * relativeTokenUpperLimit[reserveAddr[0]] /
totalTokenInVaults[0];
            }
......
}
```

[Xunion Swap-03] Parameter acquisition exception

Severity Level	High
Type	Business Security
Lines	xunionswapvaults.sol #L133-L144
Description	In the dereaseLpAmount function, the totalTokenInVaults parameter used in this function is the balance of the token in this contract, and at this point, the liquidity token has already been transfer to the caller, resulting in a smaller token balance obtained here, which will cause the calculated data to be abnormal.

```
function dereaseLpAmount(address _lp,uint[2] memory _reserveOut,uint _lpDel) external onlyLpManager{
        address[2] memory reserveAddr = getLpPair( _lp) ;
        uint[2] memory totalTokenInVaults;
        totalTokenInVaults[0] = IERC20(reserveAddr[0]).balanceOf(address(this));//getLpTokenSum( _lp);//
        totalTokenInVaults[1] = IERC20(reserveAddr[1]).balanceOf(address(this));
        require(totalTokenInVaults[0]>0&&totalTokenInVaults[1]>0,"X Swap Vaults: Vaults have NO reserve");
        reserves[_lp].reserve[0] -= _reserveOut[0] * relativeTokenUpperLimit[reserveAddr[0]]/totalTokenInVaults[0];
        reserves[_lp].reserve[1] -= _reserveOut[1] * relativeTokenUpperLimit[reserveAddr[1]]/totalTokenInVaults[1];
        relativeTokenUpperLimit[reserveAddr[0]] -= _reserveOut[0] * relativeTokenUpperLimit[reserveAddr[0]] / totalTokenInVaults[0];
        relativeTokenUpperLimit[reserveAddr[1]] -= _reserveOut[1] * relativeTokenUpperLimit[reserveAddr[1]] / totalTokenInVaults[1];
        reserves[_lp].totalSupply -= _lpDel;
    }
```

Recommendation	TotalTokenInVaults should be added with reserveOut, or the dereaseLpAmount function should be called first before transfer the token
Status	**Fixed.**

```
function dereaseLpAmount(address _lp,uint[2] memory _reserveOut,uint _lpDel) external onlyLpManager{
        address[2] memory reserveAddr = getLpPair( _lp) ;
```

```
        uint[2] memory totalTokenInVaults;
        totalTokenInVaults[0] =
IERC20(reserveAddr[0]).balanceOf(address(this)) +
_reserveOut[0];//getLpTokenSum( _lp);//
        totalTokenInVaults[1] =
IERC20(reserveAddr[1]).balanceOf(address(this)) + _reserveOut[1];
        require(totalTokenInVaults[0]>0&&totalTokenInVaults[1]>0,"X
Swap Vaults: Vaults have NO reserve");
        reserves[_lp].reserve[0] -= _reserveOut[0] *
relativeTokenUpperLimit[reserveAddr[0]]/totalTokenInVaults[0];
        reserves[_lp].reserve[1] -= _reserveOut[1] *
relativeTokenUpperLimit[reserveAddr[1]]/totalTokenInVaults[1];
        relativeTokenUpperLimit[reserveAddr[0]] -= _reserveOut[0] *
relativeTokenUpperLimit[reserveAddr[0]] / totalTokenInVaults[0];
        relativeTokenUpperLimit[reserveAddr[1]] -= _reserveOut[1] *
relativeTokenUpperLimit[reserveAddr[1]] / totalTokenInVaults[1];
        reserves[_lp].totalSupply -= _lpDel;
    }
```

[Xunion Swap-04] Price updates are not synchronized

Severity Level	High
Type	Business Security
Lines	xunionswapcore.sol #L85-L86

Description	In the `swapCalculation` function, can calculate the number of tokens that can be exchanged based on the number of tokens passed in, and calculate the new `reserve` and `priceCumulative`. However, the `priceCumulative` updates for the two tokens in the pair are not synchronized, resulting in abnormal exchange quantities when swapping again.
	`priceCumulative[j] = (1 ether+a)*_lpDetails.reserve[1-j]/1 ether - _outputAmount;`
	`priceCumulative[1-j] = _lpDetails.reserve[j] + _inputAmount;`

Recommendation	Synchronize and update two `priceCumulatives`, so that both are expanded by the order of magnitude `a`.

Status	**Fixed.**
	`priceCumulative[j] = (1 ether+a)*_lpDetails.reserve[1-j]/1 ether - _outputAmount;`
	`priceCumulative[1-j] = _lpDetails.reserve[j] + b + _inputAmount;`

[Xunion Swap-05] Incorrect use of variables

Severity Level	**Medium**
Type	Business Security
Lines	xunionswapcore.sol #L59,L115,L173
Description	The `swapCalculation` (L 26), `swapCalculation 2` (L 89), and `swapCalculation 3` (L 146) functions have an issue with incorrect variable usage. Each function defines the a variable, and when executing if else below, `a` is always 0, resulting in variables `a` and `b` always being 0, which cannot achieve business results.

```
    function swapCalculation(address _lp,address _inputToken,uint
_inputAmount,uint _i)public view returns (uint _outputAmount,uint[2]
memory reserve,uint[2] memory priceCumulative,uint b) {
......
        uint a;
......
        if(a == 0){
            b = 0;
        }else{
......
}
```

Recommendation	Using variables correctly based on business logic.
Status	**Fixed.**

```
    function swapCalculation(address _lp,address _inputToken,uint
_inputAmount,uint _i)public view returns (uint _outputAmount,uint[2]
memory reserve,uint[2] memory priceCumulative,uint b) {

......

        _lpDetails = obtainReserves(_lp);

        uint a = _lpDetails.a0;

 ......

        if(a == 0){

            b = 0;

        }else{

 ......
```

```
    }
```

[Xunion Swap-06] Swap efficiency is too low

Severity Level	Low
Type	Business Security
Lines	xunionswapvaults.sol #L215
Description	In the exchange function, restricts a block to only have one swap transaction, which may cause many users to fail their transactions.

```
function exchange(structlibrary.exVaults memory _exVaults,uint
deadline) public lock returns(uint){
        require(latestTimestamp<block.timestamp,"X Swap Vaults: Same
block can't have Two exchange");
        latestTimestamp = block.timestamp;
......
}
```

Recommendation	If it is necessary to limit the number of transactions, it is recommended to only perform one swap on a single user for the same block, and to remove the restriction on all users.
Status	**Fixed.**

```
function exchange(structlibrary.exVaults memory _exVaults,uint
deadline) public lock returns(uint){
        if(msg.sender != xInterface){
            require(latestBlockNumber < block.number,"X Swap Vaults:
Same block can't have Two exchange");
        }
        latestBlockNumber = block.number;
......
}
```

[Xunion Swap-07] No function to handle ETH transfers

Severity Level	Info
Type	Business Security
Lines	xunionswapvaults.sol
Description	This contract has `fallback` and `receive` functions that can receive ETH, but the contract does not have code to handle ETH. If a user mistakenly transfers ETH to the contract, it will result in ETH being locked in the contract and unable to be extracted.
Recommendation	It is recommended to adding a function to extract ETH, or removing the `fallback` and `receive` functions.
Status	**Fixed.**

[Xunion Swap-08] Centralization risk

Severity Level	Info
Type	Business Security
Lines	xunionswapvaults.sol
Description	The transaction fee for this contract has no upper limit, and the process of changing the fee has not triggered an event. May lead to centralization risks.
Recommendation	It is recommended to adding a maximum handling fee limit.
Status	**Fixed.**

[Xunion Swap-09] Redundant Code

Severity Level	Info
Type	Coding Conventions
Lines	xunionswapvaults.sol #L149 xunionswappair.sol #L12 xunionswapfactory.sol #L26
Description	There are multiple code redundancies in the project.

```
function addTokenApproveToLpManager(address _token) external
onlyLpManager{
        IERC20(_token).approve(lpManager,
99999999999999999999999999999999999999 ether);
    }
```

```
uint public constant MINIMUM_LIQUIDITY = 10**3;
```

```
modifier lock() {
        require(unlocked == 1, 'X SWAP Factory: LOCKED');
        unlocked = 0;
        _;
        unlocked = 1;
}
```

Recommendation	Delete redundant code
Status	**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	Medium	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.

3.1.3 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.

3.1.4 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
		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/beosin

Twitter
https://twitter.com/Beosin_com

Email
service@beosin.com