BEOSIN
Blockchain Security

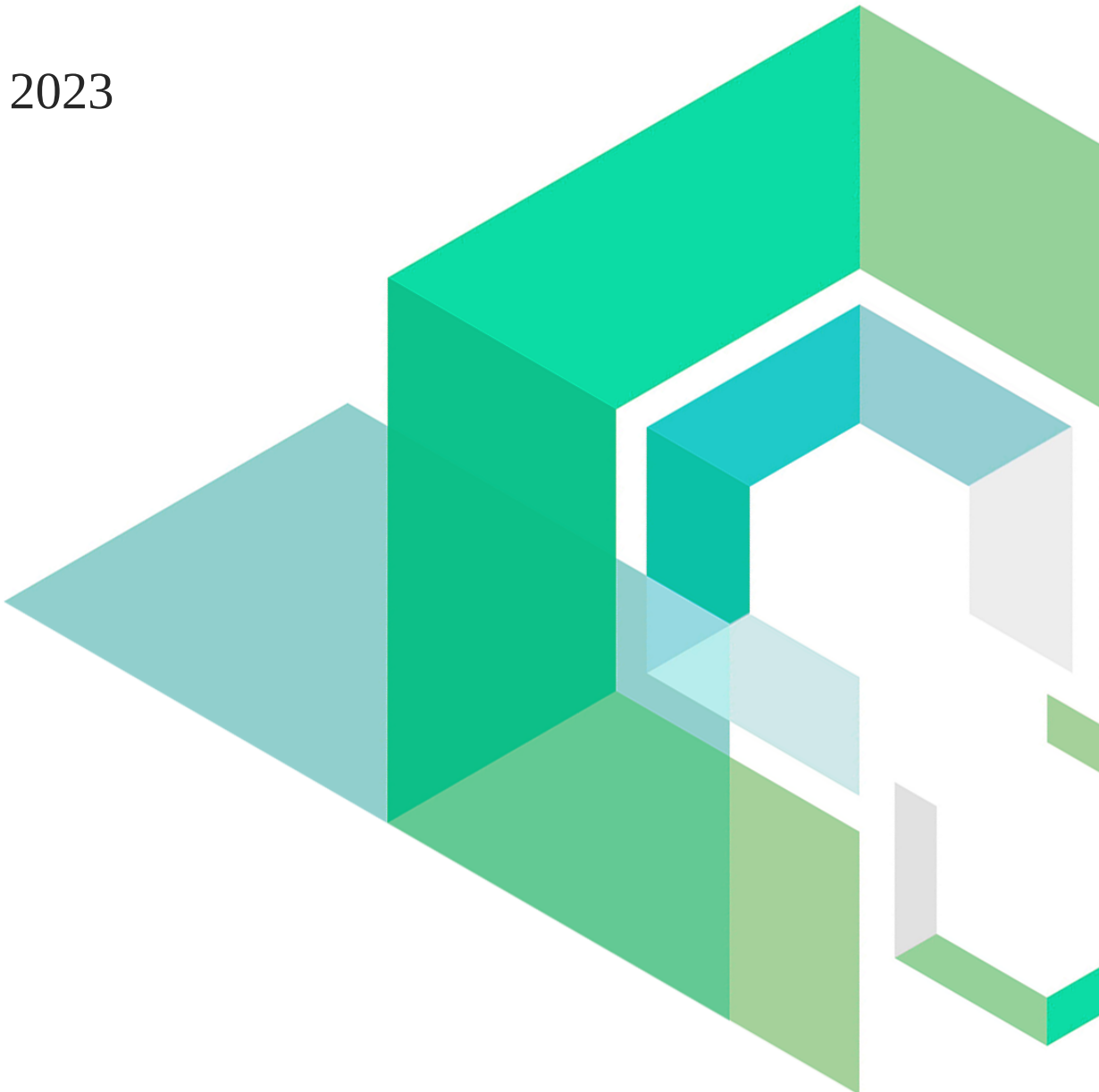
Arbswap-StableSwap

Smart Contract Security Audit

V1.0

No. 202304270945

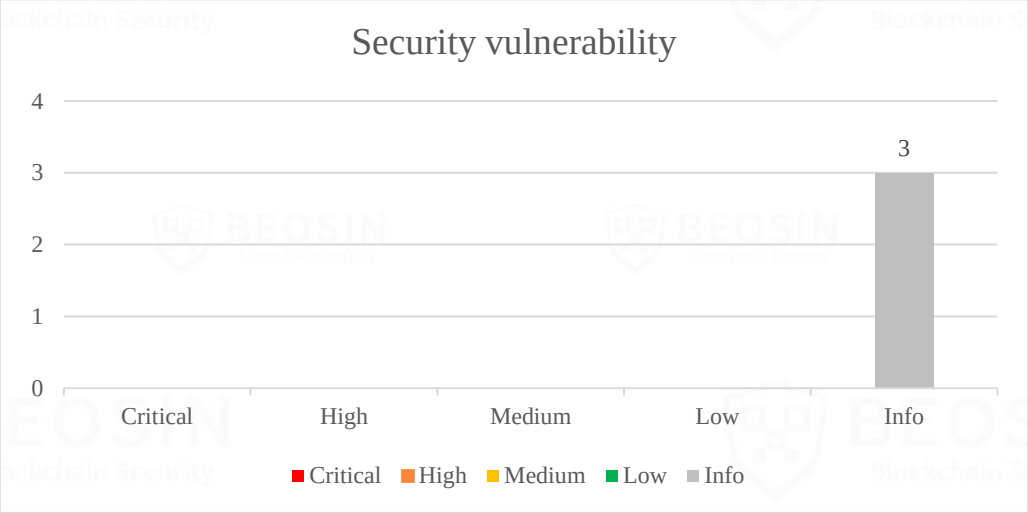
Apr 27th, 2023



Contents

Summary of Audit Results

After auditing, 3 Info-risks were identified in the Arbswap-StableSwap 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:**

  For the project side, it is recommended to follow the check-effect-interaction principle for processing, burn the LP token after transferring the user's token during removing liquidity, and minimize the gas value in the call without affecting the business, as a way to prevent known read-only re-entry issue. For other protocols' developer, when interacting directly with the pool contract, need to pay attention to the known read-only reentrancy risks, and avoid reentrancy in function implementation.

Business overview

Arbswap-StableSwap is a stablecoin trading platform where users can engage in AMM and exchange between anchor dollar assets such as USDC, USDT, DAI, and derivative assets of coins e.g. BTC, ETH, and others.

The project utilizes a stablecoin pricing algorithm similar to that of CurveFi, and the liquidity pools are similar to CurveFi's basepool and ethpool. Currently, each liquidity pool supports two to three assets, including the native token of the chain. The project uses the design pattern of the contract factory, including liquidity pools for 2pool/3pool and LPtokens, all of which are deployed by their corresponding factory contracts. After deploying the LP and pool factory contracts, the contract deployer must transfer permissions such as owner and minter to the swap factory contract to use its functions normally. The pool contracts have a pause function that can only be called by the owner and cannot be used after 60 days from the contract's deployment.

The project also implements some utils tools that provide a query interface for poolinfo and optimize the process of adding/removing liquidity involved in wrapped native tokens.

1 Overview

1.1 Project Overview

Project Name	Arbswap-StableSwap
Platform	Arbitrum
Contract Address	0x3a52e9200Ed7403D9d21664fDee540C2d02c099d (ArbswapStableSwapFactory) 0x5bbebD93777DFa334f6346271FBC6a56Ed3718F1 (ArbswapStableSwapLPFactory) 0xc1b418879750Fc785B2112a77d73498EaA125e67 (ArbswapStableSwapThreePoolDeployer) 0x6b23bBddDBbef5F999B9CB7e9b579231aC69507C (ArbswapStableSwapTwoPoolDeployer)

1.2 Audit Overview

Audit work duration: Apr 21, 2023 – Apr 27, 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
Arbswap-StableSwap-1	The value of ThreePoolInfo may be overridden	Info	Acknowledged
Arbswap-StableSwap-2	Reentrancy risk	Info	Acknowledged
Arbswap-StableSwap-3	Redundant code	Info	Acknowledged

Status Notes:

1. Arbswap-StableSwap-1 is not fixed and may cause the previously created pool information will be overwritten.

2. Arbswap-StableSwap-2 is not fixed and may lead to known read-only reentrancy risk in special cases.

3. Arbswap-StableSwap-3 is not fixed and may not cause any issue.

Finding Details:

[Arbswap-StableSwap-1] The value of ThreePoolInfo may be overridden

Severity Level	Info
Type	General Vulnerability
Lines	ArbswapStableSwapFactory.sol#L27, L127-171, L203-210
Description	When creating a 3pool, if t0 and t1 are consistent with the created pool, it will cause the value of ThreePoolInfo to be rewritten, and the previously created pool information will be overwritten.

```
26    // Query three pool pair infomation by two tokens.
27    mapping(address => mapping(address => StableSwapThreePoolPairInfo)) threePoolInfo;
```

Figure 1 Source code of related contract

```
127   function createThreePoolPair(
128       address _tokenA,
129       address _tokenB,
130       address _tokenC,
131       uint256 _A,
132       uint256 _fee,
133       uint256 _admin_fee
134   ) external onlyOwner {
135       require(
136           _tokenA != ZEROADDRESS &&
137               _tokenB != ZEROADDRESS &&
138               _tokenC != ZEROADDRESS &&
139               _tokenA != _tokenB &&
140               _tokenA != _tokenC &&
141               _tokenB != _tokenC,
142           "Illegal token"
143       );
144       (address t0, address t1, address t2) = sortTokens(_tokenA, _tokenB, _tokenC);
145       address LP = LPFactory.createSwapLP(t0, t1, t2, address(this));
146       address swapContract = SwapThreePoolDeployer.createSwapPair(t0, t1, t2, _A, _fee, _admin_fee, msg.sender, LP);
147       IArbswapStableSwapLP(LP).setMinter(swapContract);
148       addPairInfoInternal(swapContract, t0, t1, t2, LP);
149   }
150
151   function addPairInfoInternal(
152       address _swapContract,
153       address _t0,
154       address _t1,
155       address _t2,
156       address _LP
157   ) internal {
158       StableSwapThreePoolPairInfo storage info = stableSwapPairInfo[_t0][_t1][_t2];
159       info.swapContract = _swapContract;
160       info.token0 = _t0;
161       info.token1 = _t1;
162       info.token2 = _t2;
163       info.LPContract = _LP;
164       swapPairContract[pairLength] = _swapContract;
165       pairLength += 1;
166       if (_t2 != ZEROADDRESS) {
167           addThreePoolPairInfo(_t0, _t1, _t2, info);
168       }
169
170       emit NewStableSwapPair(_swapContract, _t0, _t1, _t2, _LP);
171   }
```

Figure 2 Source code of related contract

```
203     function getThreePoolPairInfo(address _tokenA, address _tokenB)
204         external
205         view
206         returns (StableSwapThreePoolPairInfo memory info)
207     {
208         (address t0, address t1) = sortTokens(_tokenA, _tokenB);
209         info = threePoolInfo[t0][t1];
210     }
```

Figure 3 Source code of related contract

Recommendations	It is recommended to use the mapping stableSwapPairInfo to query 3pool information.
Status	Acknowledged. They will avoid this problem by not to create 3pool when there're existed 2pool according to the project party.

[Arbswap-StableSwap-2] Reentrancy risk

Severity Level	**Info**
Type	General Vulnerability
Lines	ArbswapStableSwapTwoPool.sol#L468-484 ArbswapStableSwapThreePool.sol#L468-484
Description	Making an external call before burning the LP token in the *remove_liquidity* function may lead to known read-only reentrancy risk in special cases.

```
function remove_liquidity(uint256 _amount, uint256[N_COINS] memory min_amounts) external nonReentrant {
    uint256 total_supply = token.totalSupply();
    uint256[N_COINS] memory amounts;
    uint256[N_COINS] memory fees; //Fees are unused but we've got them historically in event

    for (uint256 i = 0; i < N_COINS; i++) {
        uint256 value = (balances[i] * _amount) / total_supply;
        require(value >= min_amounts[i], "Withdrawal resulted in fewer coins than expected");
        balances[i] -= value;
        amounts[i] = value;
        transfer_out(coins[i], value);
    }

    token.burnFrom(msg.sender, _amount); // dev: insufficient funds

    emit RemoveLiquidity(msg.sender, amounts, fees, total_supply - _amount);
}
```

Figure 4 Source code of *remove_liquidity* function

Recommendations	It is recommended to follow the principle of Checks-Effects-Interactions for processing, transfer out the user's token after burning the LP token, and minimize the value of gas in the call without affecting the business.
Status	Acknowledged.

[Arbswap-StableSwap-3] Redundant code

Severity Level	**Info**
Type	Coding Conventions
Lines	ArbswapStableSwapFactory.sol#L184-192
Description	This function can only be called by the owner, and only updates the value of threePoolInfo, which is not necessary.

```
184     function addPairInfo(address _swapContract) external onlyOwner {
185         IArbswapStableSwap swap = IArbswapStableSwap(_swapContract);
186         uint256 N_COINS = swap.N_COINS();
187         if (N_COINS == 2) {
188             addPairInfoInternal(_swapContract, swap.coins(0), swap.coins(1), ZEROADDRESS, swap.token());
189         } else if (N_COINS == 3) {
190             addPairInfoInternal(_swapContract, swap.coins(0), swap.coins(1), swap.coins(2), swap.token());
191         }
192     }
```

Figure 5 Source code of *addPairInfo* function

Recommendations	It is recommended to delete the redundant code.
Status	Acknowledged.

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