BEOSIN
Blockchain Security

W3Swap Multiple Reward Farm

Smart Contract Security Audit

V1.0

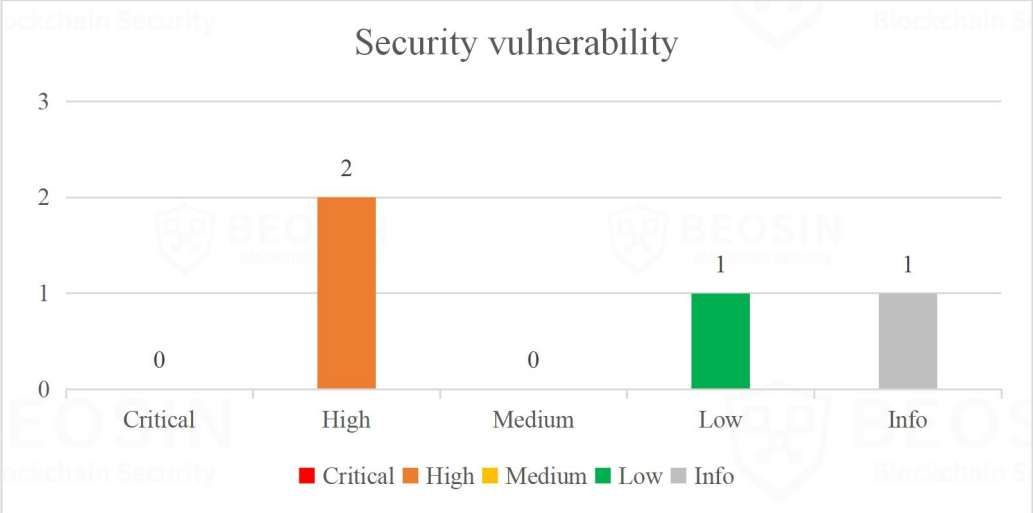
No. 202304241000

Apr 24th, 2023

Contents

Summary of Audit Results

After auditing, 2 High, 1 Low and 1 Info risk items were identified in the W3Swap Multiple Reward Farm 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. If the owner sets the reward token as the user's stake token, it will likely result in the loss of the user's principal.

- **Project Description:**

1. Business overview

The W3Swap-Multiple-Reward-Farm project is a smart contract similar to MasterChef, but compared to MasterChef, it provides additional features that allow users to stake a liquidity pool (LP) and receive multiple reward tokens. Among them, masterchef.sol is a multi-mining farm on the PG chain (stake one token and get multiple reward tokens), which provides users with the corresponding number of w3 credential token when they stake and destroys the corresponding number of w3 credential token when they withdraw. mastercheftInit and masterchefBsc are multi-mining farms that do not provide The mastercheftInit.sol and masterchefBsc.sol are multi-dig farms that do not provide credential token.

Specifically, the contract achieves its functionality through the following:

- Manages the asset pools: the contract monitors the asset pools of the different pairs in the protocol and records the number and value of LPs in each asset pool.
- Issues token rewards: the contract calculates the token rewards that each liquidity provider should receive according to certain rules.
- Control the speed of rewards: Contracts can set different reward speeds to control the total amount and timing of tokens issued. This helps prevent excessive devaluation or over-dilution of funds.
- Provides voucher tokens: when users stake their tokens, the contract provides a corresponding number of w3sp tokens as voucher tokens.

1 Overview

1.1 Project Overview

Project Name	W3Swap Multiple Reward Farm
Platform	EVM Compatible Chains
Audit scope	masterchef.sol 3dbcfe8ec9c8285e471298287761dbfbc00e6743d6a114890e4412a668fd7930 masterchefBsc.sol 559c9ccf0f739377726d477293ef43c8055ebd5c016f87cfc137956847ec9373 mastercheftInit.sol 2b7ba4da746b0fdbd68d9050713de8bce16350b078d2e0d87cc4a44b4acc2224

1.2 Audit Overview

Audit work duration: Apr 21, 2023 – Apr 24, 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
W3Swap Multiple Reward Farm-1	The user's principal cannot be withdrawn	High	Fixed
W3Swap Multiple Reward Farm-2	The *withdrawSafe* function defect	High	Fixed
W3Swap Multiple Reward Farm-3	Centralization Risk	Low	Acknowledged
W3Swap Multiple Reward Farm-4	Lack of event triggering	Info	Acknowledged

Status Notes:

1. W3Swap Multiple Reward Farm-3 is not fixed and if the owner parameters are not set properly, it will lead to the loss of user funds.

2. W3Swap Multiple Reward Farm-4 is not fixed and does not pose any risks.

Finding Details:

[W3Swap Multiple Reward Farm-1] The user's principal cannot be withdrawn

Severity Level	High
Type	Business Security
Lines	MasterChef.sol #L304-307
Description	The user can stake LPToken to the contract and the FtToken token contract will mint corresponding number of tokens for the user. However, there is an issue with the transferFrom function in the FtToken contract, which allows users to call transferFrom function without authorization to transfer tokens belonging to other users to a whiteAddress. This creates a vulnerability where attackers can transfer FtToken tokens held by users to the whiteAddress, thus preventing them from calling the withdraw function to retrieve their assets.

```
304  function transferFrom(address sender, address recipient, uint256 amount) public virtual override returns (bool) {
305      if(whiteAddress == recipient){
306          _transfer(sender, recipient, amount);
307      }else{
308          _transfer(sender, recipient, amount);
309          _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "BEP40: transfer amount exceeds allowance"));
310      }
311      return true;
312  }
```

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

```
548      function deposit(uint256 _pId,uint256 _amount) public{
549          require(hasPoolExist(_pId),'Pool not Exist');
550          if(poolInfo[_pId].tokenInfos.length > 0){
551              updatePoolAllToken(_pId,msg.sender);
552              poolInfo[_pId].totalSupply = poolInfo[_pId].totalSupply.add(_amount);
553              userInfo[_pId][msg.sender].amount = userInfo[_pId][msg.sender].amount.add(_amount);
554              poolInfo[_pId].lpToken.safeTransferFrom(msg.sender, address(this), _amount);
555              poolInfo[_pId].ftToken.mint(msg.sender,_amount);
556              emit Deposited(_pId,_amount,msg.sender);
557          }
558      }
559
560      function withdraw(uint256 _pId,uint256 _amount) public{
561          require(hasPoolExist(_pId),'Pool not Exist');
562          require(userInfo[_pId][msg.sender].amount >= _amount,'Amount over');
563          updatePoolAllToken(_pId,msg.sender);
564          poolInfo[_pId].ftToken.burn(msg.sender,_amount);
565          poolInfo[_pId].lpToken.safeTransfer(msg.sender, _amount);
566          poolInfo[_pId].totalSupply = poolInfo[_pId].totalSupply.sub(_amount);
567          userInfo[_pId][msg.sender].amount = userInfo[_pId][msg.sender].amount.sub(_amount);
568          emit Withdrawed(_pId,_amount,msg.sender);
569      }
```

Figure 2 Source code of related functions

Recommendations	It is recommended to remove the logic that the *transferFrom* function in the FtToken token contract does not require authorization to transfer tokens to the whiteAddress address to avoid potential attack risks.
Status	Fixed.

```
296
297  function transferFrom(address sender, address recipient, uint256 amount) public virtual override returns (bool) {
298      _transfer(sender, recipient, amount);
299      _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "BEP40: transfer amount exceeds allowance"));
300      return true;
301  }
```

Figure 3 Source code of *transferFrom* function (fixed)

[W3Swap Multiple Reward Farm-2] The *withdrawSafe* function defect

Severity Level	High
Type	Business Security
Lines	MasterChef.sol #L571-580
Description	The user will not be able to withdraw the funds through the withdrawSafe function because the user's book is emptied first in the *withdrawSafe* function, resulting in a zero amount of LP transferred to the user by the contract, and the w3 token held by the user are not destroyed when they are withdrawn through the emergency.

```
571  function withdrawSafe(uint256 _pId) public{
572      require(hasPoolExist(_pId),'Pool not Exist');
573      if(safeWithdraw){
574          userInfo[_pId][msg.sender].amount = 0;
575          for(uint256 k=0;k<poolInfo[_pId].tokenInfos.length;k++){
576              poolInfo[_pId].tokenInfos[k].rewards[msg.sender] = 0;
577          }
578          poolInfo[_pId].lpToken.safeTransfer(msg.sender, userInfo[_pId][msg.sender].amount);
579      }
580  }
```

Figure 4 Source code of *withdrawSafe* function (unfixed)

Recommendations	It is recommended to start with a temporary variable to store the number of tokens staked by the user. Also destroy the user's corresponding number of voucher token.
Status	Fixed.

```
553  function withdrawSafe(uint256 _pId) public{
554      require(hasPoolExist(_pId),'Pool not Exist');
555      if(safeWithdraw){
556          for(uint256 k=0;k<poolInfo[_pId].tokenInfos.length;k++){
557              poolInfo[_pId].tokenInfos[k].rewards[msg.sender] = 0;
558          }
559          poolInfo[_pId].lpToken.safeTransfer(msg.sender, userInfo[_pId][msg.sender].amount);
560          userInfo[_pId][msg.sender].amount = 0;
561      }
562  }
```

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

[W3Swap Multiple Reward Farm-3] Centralization Risk

Severity Level	Low
Type	Business Security
Lines	MasterChef.sol #L458-512
Description	The function that adds reward tokens to the contract does not check if the staked tokens cannot be used as reward tokens, which may result in the user's capital being at risk. For instance, suppose Pool 1 has token A as staked token and token B as reward token. If the administrator mistakenly sets token A of Pool 1 as the reward token for Pool 2, users of Pool 1 will lose their capital and it will be awarded to Pool 2 users if the rewards are insufficient. The functions addPool, addTokenInPool, and addTokenInPoolMulti can all set the user's staked tokens as reward tokens.

```
458    function addPool(uint256 _pId,address _lpAddress,address _tokenAddress,uint256 _poolWeight) public onlyOwner{
459        require(!hasPoolExist(_pId),'Pool has Exist');
460        updateAllPool(address(0));
461        W3SP w3sp = new W3SP();
462        w3sp.setWhiteAddress(whiteAddress);
463        w3sp.mint(msg.sender,1000);
464        poolInfoSize = poolInfoSize.add(1);
465        poolInfo[_pId].isAdd = true;
466        poolInfo[_pId].lpToken = IERC20(_lpAddress);
467        poolInfo[_pId].ftToken = w3sp;
468        poolInfo[_pId].ftTokenAddress = address(w3sp);
469        ftToLpMapping[address(w3sp)] = _lpAddress;
470        TokenInfo memory t;
471        t.token = IERC20(_tokenAddress);
472        t.tokenAddress = _tokenAddress;
473        t.poolWeight = _poolWeight;
474        poolInfo[_pId].tokenInfos.push(t);
475        allTokenWeight[_tokenAddress] = allTokenWeight[_tokenAddress].add(_poolWeight);
476        emit PoolAdded(_pId,_lpAddress);
477        emit TokenAdded(_pId,_tokenAddress,_poolWeight);
478        emit LpCreated(_lpAddress,address(w3sp));
479    }
480
481    function addTokenInPool(uint256 _pId,address _tokenAddress,uint256 _poolWeight) public onlyOwner{
482        require(hasPoolExist(_pId),'Pool not Exist');
483        updateAllPool(address(0));
484        TokenInfo memory t;
485        t.token = IERC20(_tokenAddress);
486        t.tokenAddress = _tokenAddress;
487        t.poolWeight = _poolWeight;
488        t.lastUpdateBlock = block.number;
489        poolInfo[_pId].tokenInfos.push(t);
490        allTokenWeight[_tokenAddress] = allTokenWeight[_tokenAddress].add(_poolWeight);
491        emit TokenAdded(_pId,_tokenAddress,_poolWeight);
492    }
```

Figure 6 Source code of related functions (unfixed)

```
494    function addTokenInPoolMulti(uint256[] memory _pIds,address[] memory _tokenAddresss,uint256[] memory _poolWeights) public onlyOwner{
495        require(_pIds.length == _tokenAddresss.length,'error length');
496        require(_pIds.length == _poolWeights.length,'error length');
497        updateAllPool(address(0));
498        for(uint256 i=0;i<_pIds.length;i++){
499            uint256 _pId = _pIds[i];
500            address _tokenAddress = _tokenAddresss[i];
501            uint256 _poolWeight = _poolWeights[i];
502            if(hasPoolExist(_pId)){
503                TokenInfo memory t;
504                t.token = IERC20(_tokenAddress);
505                t.tokenAddress = _tokenAddress;
506                t.poolWeight = _poolWeight;
507                t.lastUpdateBlock = block.number;
508                poolInfo[_pId].tokenInfos.push(t);
509                allTokenWeight[_tokenAddress] = allTokenWeight[_tokenAddress].add(_poolWeight);
510            }
511        }
512        emit TokenAddedMulti(_pIds,_tokenAddresss,_poolWeights);
513    }
```

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

Recommendations	It is recommended that the function determine that the reward tokens cannot be LP tokens staked by the user.
Status	Acknowledged.

[W3Swap Multiple Reward Farm-4] Lack of event triggering	
Severity Level	Info
Type	Coding Conventions
Lines	Aridrop.sol #L427-432
Description	Special role operations are not logged.

```
426
427     function setWhiteAddress(address whiteAddress_) public onlyOwner{
428         whiteAddress = whiteAddress_;
429     }
430
431     function setSafeWithdraw(bool _start) public onlyOwner{
432         safeWithdraw = _start;
433     }
```

Figure 8 Source code of related functions (unfixed)

Recommendations	It is recommended to add corresponding events and trigger them.
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	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/+dD8Bnqd133RmNWNl

Twitter

https://twitter.com/Beosin_com

Email

Contact@beosin.com