BEOSIN
Blockchain Security

W3launchpad

Smart Contract Security Audit

V1.1

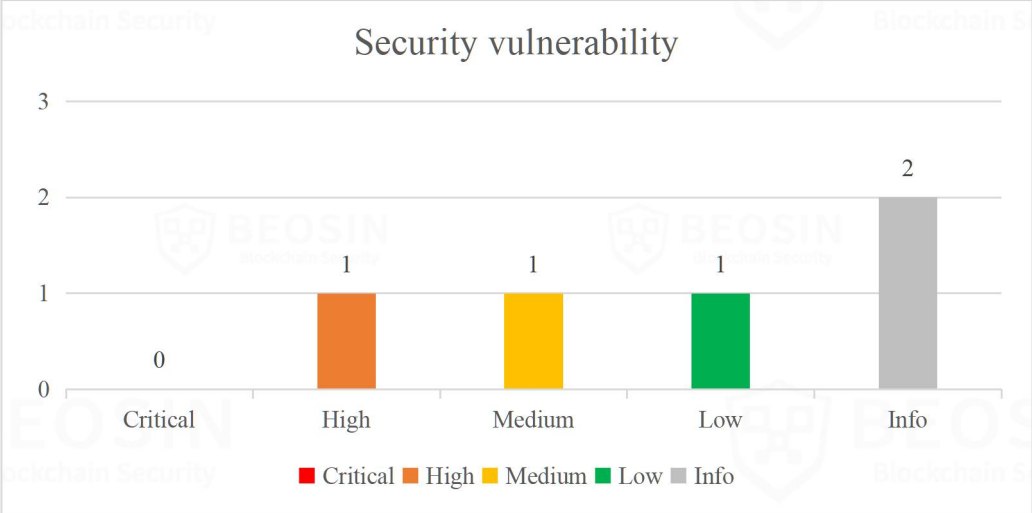
No. 202305301830

May 30th, 2023

Contents

Summary of Audit Results

After auditing, 1 High-risk, 1 Medium-risk, 1 Low-risk and 2 Info items were identified in the W3launchpad 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:**

1. When the token issuer calls the *finalize* function, the funds used to add liquidity will be transferred to the project party if the auto-listing condition is met and the liquidity pool already exists at that time. It is recommended that token issuers pay attention to the configuration of the parameters.

2. It is recommended that token issuers use standard BEP-20 token.

3. It is recommended that currencyToken only use tokens with a precision of 18, otherwise the project will run with errors.

- **Project Description:**

1. Business overview

W3launchpad is a token pre-sale program that allows token issuers to pre-sale their tokens through the W3Launchpad contract, which creates a W3PoolNormal contract for the issuer and locks the issuer's pre-sale tokens into it. Once the pre-sale has started, users can purchase launch tokens with tokens specified by the issuer, but will not be able to collect them until the pre-sale has ended.

The issuer can decide to end the pre-sale at any time. When the pre-sale ends, the funds raised and the remaining pre-sale tokens in the contract will be sent back to the issuer. A fee will be charged by the project owner for this process. If the issuer has preset an automatic listing and the amount raised reaches the expected value, the contract will automatically create a liquidity pool and add liquidity at the rate initially set by the issuer and lock the liquidity into the W3Lock contract. However, if the liquidity pool already exists, the contract will send the funds used to add liquidity to the project owner.

At the end of the pre-sale, users can withdraw their purchased tokens or, if the issuer has set up a batch release, it will take some time for them to collect all their tokens. Users can also withdraw their principal in an emergency during the pre-sale, although a fee will be charged.

1 Overview

1.1 Project Overview

Project Name	W3launchpad
Platform	BNB Chain
Audit Scope	launchpad.sol
File Hash	AAFC8171FCEFB7C3C9349FB9EA24C4490208504A2C13542A03C821AD5DC485A5 ACE79133A8EA17BFDAF6EB02C32A4E6CEE4D85B016EEB56AA3A5406520A3D470 14B45FACAB89F9F30376B870A7B542982BF18741D2872812CE838679945DD838 C135E76C5F230878F73C54F11791C0B0713A85EE892E071DC4A2E29E265082D7

1.2 Audit Overview

Audit work duration: May 25, 2023 – May 30, 2023

Update time: Jun 15, 2023

Update content: Fix an issue where the last user could not withdraw money in the case of over-collection.

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
W3launchpad-1	Update Reward Ledger Exception	High	Fixed
W3launchpad-2	Some Tokens May Be Locked Into Contracts	Medium	Fixed
W3launchpad-3	contribute Function Logic Defects	Low	Fixed
W3launchpad-4	Redundant Code	Info	Fixed
W3launchpad-5	Missing Events	Info	Acknowledged

Status Notes:

1. W3launchpad-5 is unfixed, but there is no harm done.

Finding Details:

[W3launchpad-1] Update Reward Ledger Exception

Severity Level	High
Type	Business Security
Lines	launchpad.sol #L1219, L1269
Description	As the users[msg.sender].reward ledger is updated by overwriting, this will potentially result in users receiving unlimited pre-sale tokens. For example, the first time the user receives the full reward normally, the second time the user receives the reward again users[msg.sender].reward will be assigned a value of 0, and the third time the user can receive the reward again. In this way, the odd number of claims will allow users to re-claim the pre-sale tokens once.

```
1186    function claimNormal() private{
1187        //最终用户奖励是多少个token
1188        uint256 amountForUser = users[msg.sender].amount.mul(price).div(1e18);
1189        if(useVesting){
1190            uint256 tgeReleaseAmount = FullMath.mulDiv(
1191                amountForUser,
1192                firstRelease,
1193                baseRate
1194            );
1195
1196            uint256 cycleReleaseAmount = FullMath.mulDiv(
1197                amountForUser,
1198                cycleBps,
1199                baseRate
1200            );
1201
1202            uint256 currentTotal = 0;
1203            if (block.timestamp >= finalizeTimeStamp) {
1204                currentTotal =
1205                (((block.timestamp - finalizeTimeStamp) / cycle) *
1206                cycleReleaseAmount) +
1207                tgeReleaseAmount;
1208            }
1209            if (currentTotal > amountForUser) {
1210                currentTotal = amountForUser;
1211            }
1212            uint256 totalAmount = currentTotal.sub(users[msg.sender].reward);
1213            IERC20(token).safeTransfer(msg.sender, totalAmount);
1214            users[msg.sender].reward = users[msg.sender].reward.add(totalAmount);
1215
1216        }else{
1217            uint256 totalAmount = amountForUser.sub(users[msg.sender].reward);
1218            IERC20(token).safeTransfer(msg.sender, totalAmount);
1219            users[msg.sender].reward = totalAmount;
1220        }
1221    }
```

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

Recommendations	It is recommended that users[msg.sender].reward = totalAmount; be changed to users[msg.sender].reward = users[msg.sender].reward.add(totalAmount).
Status	Fixed.

```
        uint256 totalAmount = currentTotal.sub(users[msg.sender].reward);
        IERC20(token).safeTransfer(msg.sender, totalAmount);
        users[msg.sender].reward = users[msg.sender].reward.add(totalAmount);

    }else{
        uint256 totalAmount = amountForUser.sub(users[msg.sender].reward);
        IERC20(token).safeTransfer(msg.sender, totalAmount);
        users[msg.sender].reward = users[msg.sender].reward.add(totalAmount);
    }
```

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

[W3launchpad-2] Some Tokens May Be Locked Into Contracts

Severity Level	Medium
Type	Business Security
Lines	launchpad.sol #L1078
Description	If the listing conditions are met, the number of liquidityAmountToken is deducted from the variable totalCount (which records the number of tokens to be returned to the owner). If the liquidity pool does not exist, this portion of the token will be added to the liquidity pool, but if the liquidity pool already exists, this portion of the token will not be processed and will remain in the contract and cannot be withdrawn.

```
1076            if(autoListing && amountCap>= softcap){
1077                cap = _addLiquityListing(cap);
1078                totalCount = totalCount.sub(liquityAmountToken);
1079            }
```

```
1125                IERC20(token2).approve(address(uniswapRouter), amountCurrecy);
1126                uniswapRouter.addLiquidity(
1127                    token,
1128                    token2,
1129                    liquityAmountToken,
1130                    amountCurrecy,
1131                    0, // slippage is unavoidable
1132                    0, // slippage is unavoidable
1133                    address(this),
1134                    block.timestamp + 300 // Deadline: 5 minutes from now
1135                );
1136            }
1137            //锁仓
1138            uint256 balanceLp = IERC20(liquidityPool).balanceOf(address(this));
1139            IERC20(liquidityPool).approve(w3Lock, balanceLp);
1140            IW3Lock(w3Lock).lock(owner,liquidityPool,true,balanceLp,block.timestamp+liquityLock,description);
1141        }else{
1142            //如果已经有池子了，则把金额返回给手续费地址
1143            if(isBNB){
1144                serviceFeeReceiver.transfer(amountCurrecy);
1145            }else{
1146                IERC20(tokenCurrency).safeTransfer(serviceFeeReceiver, amountCurrecy);
1147            }
1148        }
```

Figure 3 Source code of related code (unfixed)

Recommendations	It is recommended to check that the *_addLiquityListing* function logic and the totalCount variable are updated correctly.
Status	Fixed. If the liquidity pool already exists, transfer this part of the token to the serviceFeeReceiver address.

```
1133        }else{
1134            //如果已经有池子了，则把金额返回给手续费地址
1135            IERC20(token).safeTransfer(serviceFeeReceiver, liquityAmountToken);
1136            if(isBNB){
1137                serviceFeeReceiver.transfer(amountCurrecy);
1138            }else{
1139                IERC20(tokenCurrency).safeTransfer(serviceFeeReceiver, amountCurrecy);
1140            }
```

Figure 4 Source code of related code (fixed)

[W3launchpad-3] *contribute* Function Logic Defects

Severity Level	Low
Type	Business Security
Lines	launchpad.sol #L1028
Description	The restriction logic is wrong so that the number of tokens sold never reaches hardcap. In the function *contribute* for selling tokens, the code checks that hardcap (the hardcap of tokens bought) - amountCap (the current number of tokens sold) - amount (the number of this purchase) >= minAmount (the minimum number of tokens bought). The intention is to prevent the remaining quantity from being too small to sell. However, the restriction logic is not perfect, so that if there is less than 2*minAount left, the token cannot be sold.

```
1025        if(!isFairLaunch){
1026            require(amountCap.add(amount) <= hardcap,'is max');
1027            //如果还剩的额度不够最小额度的金额，则不让他进行申购
1028            require(hardcap.sub(amountCap).sub(amount) >= minAmount,"amount is must more than minAmount");
1029            require(amount >= minAmount && amount <= maxAmount,'amount exceeded');
1030        }else{
1031            require(amount <= maxAmount,'amount is over');
1032        }
```

Figure 5 Source code of related code (unfixed)

Recommendations	It is recommended that the logic be modified so that the number of sells can reach hardcap.
Status	Fixed. Add logic: If this one can be sold out, the above check will not be performed.

```
1025        if(!isFairLaunch){
1026            require(amountCap.add(amount) <= hardcap,'is max');
1027            require(amount >= minAmount && amount <= maxAmount,'amount exceeded');
1028            //如果本次申购不能填充满硬顶，则进行判断如果还剩的额度不够最小额度的金额，则不让他进行申购
1029            if(amount.add(amountCap) != hardcap){
1030                require(hardcap.sub(amountCap).sub(amount) >= minAmount,"amount is must more than minAmount");
1031            }
1032        }else{
1033            require(amount <= maxAmount,'amount is over');
1034        }
```

Figure 6 Source code of related code (fixed)

[W3launchpad-4] Redundant Code

Severity Level	Info
Type	Coding Conventions
Lines	launchpad.sol #L1431, launchpad.sol #L1009-L1017
Description	As shown below, in the *getFairLaunchUserAmount* function, the parameter user is not used and the variable returnAmount is not used.

```
1008    //获取超募情况下用户实际的总奖励金额
1009    function getFairLaunchUserAmount(address user) public view returns(uint256){
1010        if(fairLaunchComplete && isFairLaunch){
1011            uint256 userMaxAmount = users[msg.sender].amount.mul(softcap).div(amountCap);
1012            uint256 returnAmount = users[msg.sender].amount.sub(userMaxAmount);
1013            uint256 amountForUser = userMaxAmount.mul(price).div(1e18);
1014            return amountForUser;
1015        }
1016        return 0;
1017    }
```

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

The token parameter in the *getPoolsAtIndex* function is not used.

```
function getPoolsAtIndex(
    address token,
    uint256 start,
    uint256 end
) public view returns (Pool[] memory) {
    uint256 length = end - start + 1;
    Pool[] memory pools = new Pool[](length);
    uint256 currentIndex = 0;
    for (uint256 i = start; i <= end; i++) {
        pools[currentIndex] = getPoolAt(i);
        currentIndex++;
    }
    return pools;
}
```

Figure 8 Source code of *getPoolsAtIndex* function (unfixed)

Recommendations	It is recommended that the above redundant code be removed.
Status	Fixed. The *getFairLaunchUserAmount* function has been removed.

```
function getPoolsAtIndex(
    uint256 start,
    uint256 end
) public view returns (Pool[] memory) {
    uint256 length = end - start + 1;
    Pool[] memory pools = new Pool[](length);
    uint256 currentIndex = 0;
    for (uint256 i = start; i <= end; i++) {
        pools[currentIndex] = getPoolAt(i);
        currentIndex++;
    }
    return pools;
}
```

Figure 9 Source code of *getPoolsAtIndex* function (fixed)

[W3launchpad-5] Missing Events

Severity Level	Info
Type	Coding Conventions
Lines	launchpad.sol #L998-L1002, #L1004-L1006, #L1062-L1097, #L1330-L1332
Description	The owner related privileged function does not contain an event trigger, which will result in the offline DApp not being able to record the related transaction via an event when the related transaction is executed.

```
function setWhiteListUser(address[] memory users) public onlyOwner{
    for(uint i=0;i<users.length;i++){
        whiteList[users[i]] = true;
    }
}

function setNeedWhiteList(bool isWhiteList) public onlyOwner{
    needWhiteList = isWhiteList;
}
```

```
function setFeeValue(uint256 fee) public onlyOwner{
    feeValue = fee;
}
```

```
function finalize() public onlyOwner{
    require(!isEnd,'is Finish');
    uint256 cap = amountCap;
    if(isFairLaunch && cap > softcap){
        cap = softcap;
        fairLaunchComplete = true;
    }

    //总共发放的token数量
    uint256 tokenCount = price.mul(cap).div(1e18);

    //剩余的token数量
    uint256 totalCount = IERC20(token).balanceOf(address(this)).sub(tokenCount);

    if(autoListing && amountCap>= softcap){
        cap = _addLiquityListing(cap);
        totalCount = totalCount.sub(liquityAmountToken);
    }
```

Figure 10 Source code of related code (unfixed)

Recommendations	It is recommended that when key operations and important variables of a contract are changed, the corresponding event is triggered so that the DApp can record it.
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.

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
		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