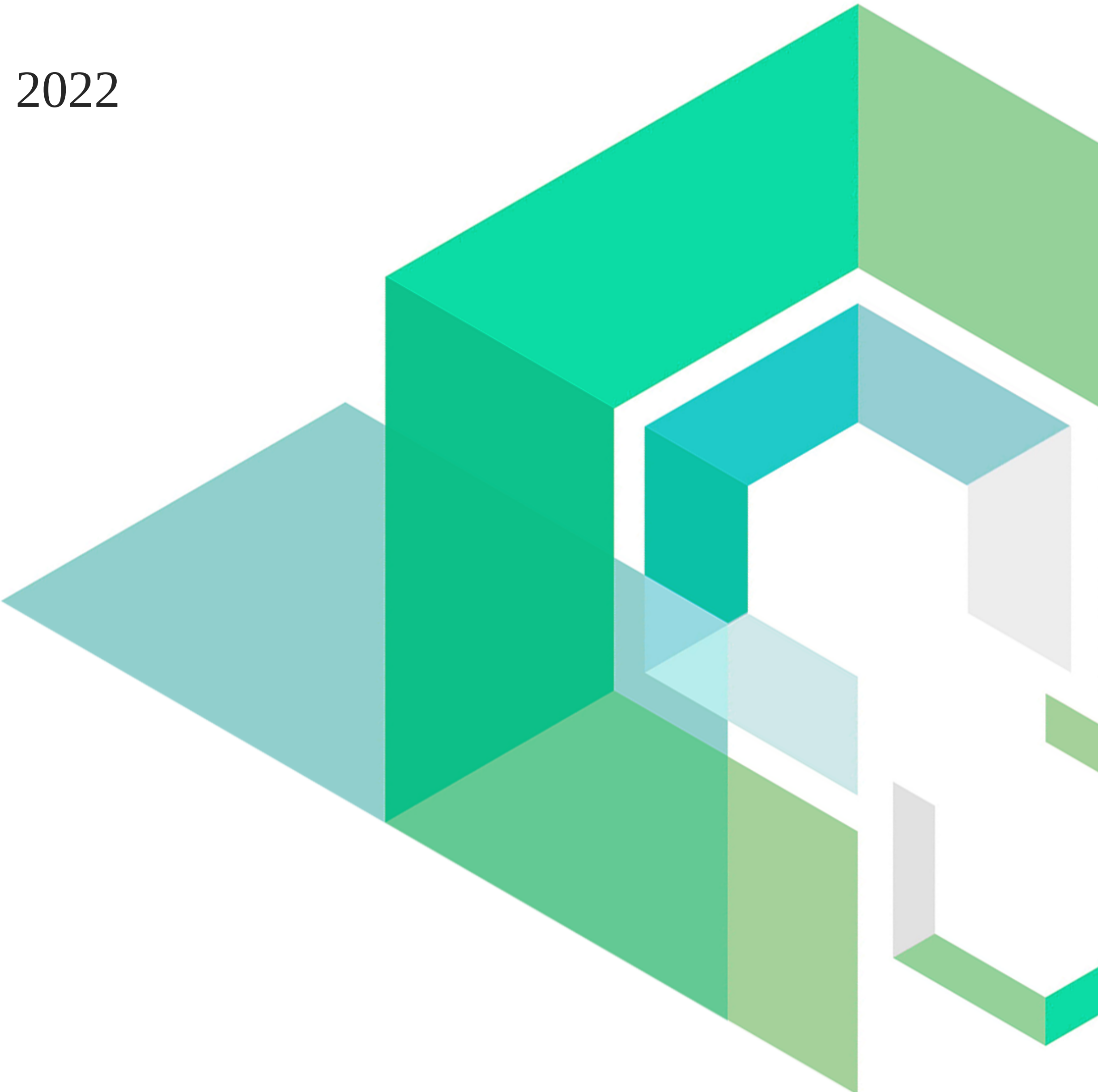
BaddeeBakers

Smart Contract Security Audit

V1.0

No. 202211301330

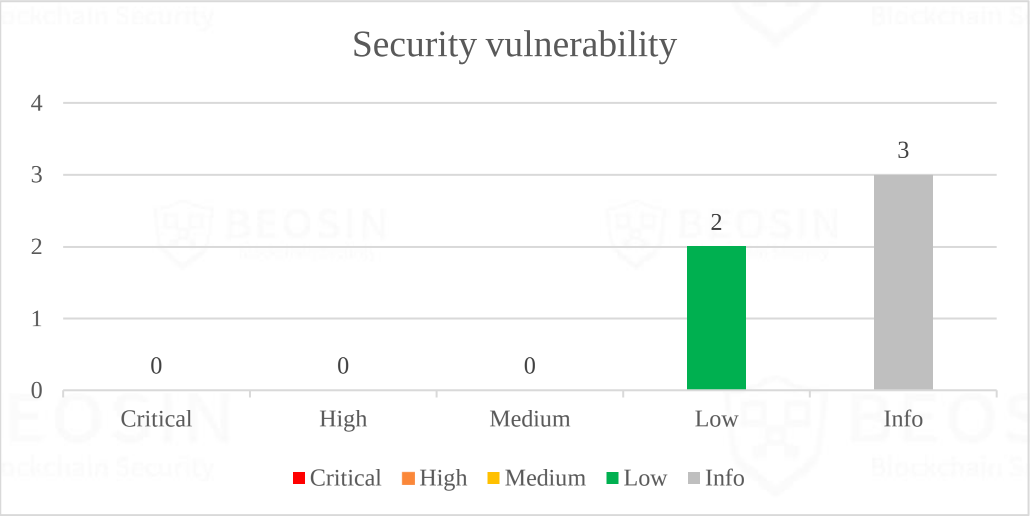
Nov 30th, 2022



Contents

Summary of Audit Results

After auditing, 2 Low-risk items and 3 Info items were identified in the BaddeeBakers project. Specific audit details will be presented in the **Findings** section. Users should pay attention to the following aspects when interacting with this project：



- **Project Description:**

1. Business overview

BaddeeBakers is an NFT project with a total of 8,888 issuance. Users can get NFT through *earlySupporterMint*, *whitelistMint* and *publicMint*. *earlySupporterMint* and *whitelistMint* require users to be in the whitelist to mint, but *publicMint* does not.

1 Overview

1.1 Project Overview

Project Name	BaddeeBakers
Platform	Ethereum
File Hash (SHA256)	DA775C268E69BF11F6D18852AC268B21A61B87DC0321EB07F313A60368817E3F (Initial) 07A747943C64F16C206E6653B3FCF2E66E278333F5F8CDA398948A852E1F3B22 (Latest)

1.2 Audit Overview

Audit work duration：November 28, 2022 – November 30, 2022

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
BaddeeBakers-1	The totalQuantity can be modified at will	Low	Fixed
BaddeeBakers-2	The key function lacks the “payable”, causing the contract to fail to compile	Low	Fixed
BaddeeBakers-3	Excess ETH not returned to users	Info	Fixed
BaddeeBakers-4	The start time should be less than the end time	Info	Fixed
BaddeeBakers-5	The set function missing event	Info	Acknowledged

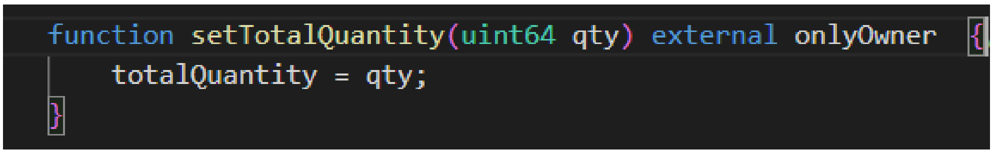
Status Notes:

- BaddeeBakers-5 is unfixed and will not cause any risk.

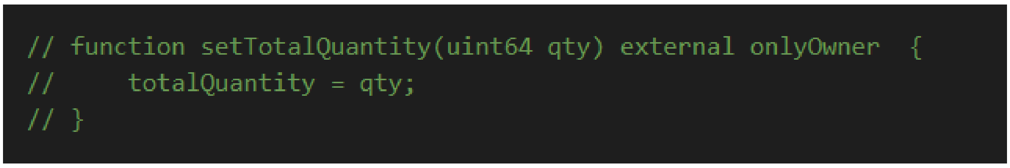
Finding Details:

[BaddeeBakers-1] The totalQuantity can be modified at will

Severity Level	**Low**
Type	Business Security
Lines	BaddeeBakers.sol #L172-L174

Description	The totalQuantity is the maximum supply and should not be modified after the contract is deployed. For example, if the modification is smaller than the original value, the user may not be able to mint.
	
	`function setTotalQuantity(uint64 qty) external onlyOwner {`
	`    totalQuantity = qty;`
	`}`
	Figure 1 The source code of *setTotalQuantity* function(Unfixed)

Recommendations	It is recommended the value of totalQuantity cannot be modified after the contract is deployed.

Status	Fixed.
	
	`// function setTotalQuantity(uint64 qty) external onlyOwner  {`
	`//     totalQuantity = qty;`
	`// }`
	Figure 2 The source code of *setTotalQuantity* function(Fixed)

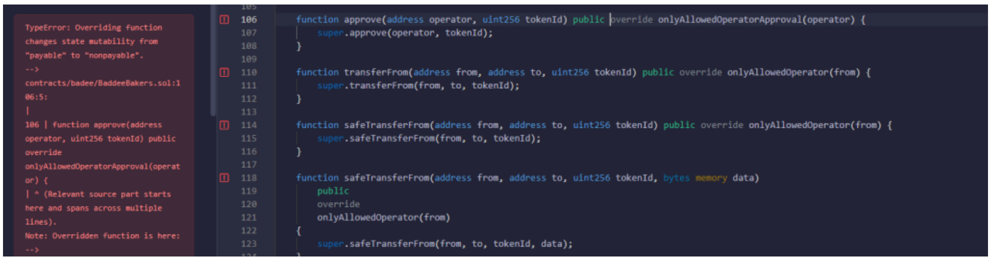
[BaddeeBakers-2] The key function lacks the “payable”, causing the contract to fail to compile

Severity Level Low

Type General Vulnerability

Lines BaddeeBakers.sol #L106-L124

Description The key function lacks the “payable”, causing the contract to fail to compile.



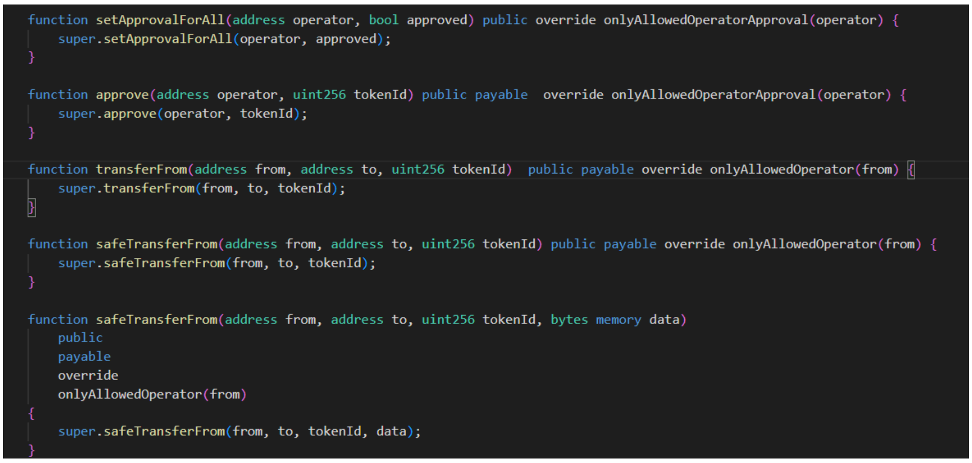
```
TypeError: Overriding function
changes state mutability from
"payable" to "nonpayable".
-->
contracts/badee/BaddeeBakers.sol:1
06:5:
|
106 | function approve(address
operator, uint256 tokenId) public
override
onlyAllowedOperatorApproval(operat
or) {
| ^ (Relevant source part starts
here and spans across multiple
lines).
Note: Overridden function is here:
-->
```

```
106  function approve(address operator, uint256 tokenId) public override onlyAllowedOperatorApproval(operator) {
107      super.approve(operator, tokenId);
108  }
109
110  function transferFrom(address from, address to, uint256 tokenId) public override onlyAllowedOperator(from) {
111      super.transferFrom(from, to, tokenId);
112  }
113
114  function safeTransferFrom(address from, address to, uint256 tokenId) public override onlyAllowedOperator(from) {
115      super.safeTransferFrom(from, to, tokenId);
116  }
117
118  function safeTransferFrom(address from, address to, uint256 tokenId, bytes memory data)
119      public
120      override
121      onlyAllowedOperator(from)
122  {
123      super.safeTransferFrom(from, to, tokenId, data);
```

Figure 3 The source code of *approve*, *transferFrom*, *safeTransferFrom* functions(Unfixed)

Recommendations It is recommended to add payable to *approve*, *transferFrom*, *safetransferFrom* functions.

Status Fixed.

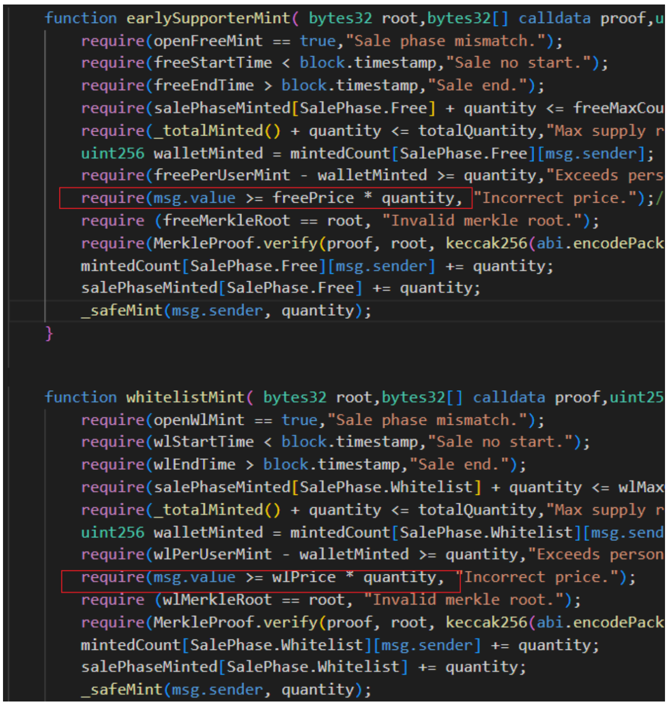


```
function setApprovalForAll(address operator, bool approved) public override onlyAllowedOperatorApproval(operator) {
    super.setApprovalForAll(operator, approved);
}

function approve(address operator, uint256 tokenId) public payable  override onlyAllowedOperatorApproval(operator) {
    super.approve(operator, tokenId);
}

function transferFrom(address from, address to, uint256 tokenId)  public payable override onlyAllowedOperator(from) {
    super.transferFrom(from, to, tokenId);
}

function safeTransferFrom(address from, address to, uint256 tokenId) public payable override onlyAllowedOperator(from) {
    super.safeTransferFrom(from, to, tokenId);
}

function safeTransferFrom(address from, address to, uint256 tokenId, bytes memory data)
    public
    payable
    override
    onlyAllowedOperator(from)
{
    super.safeTransferFrom(from, to, tokenId, data);
}
```

Figure 4 The source code of *approve, transferFrom, safeTransferFrom* functions(Fixed)

[BaddeeBakers-3] Excess ETH not returned to users

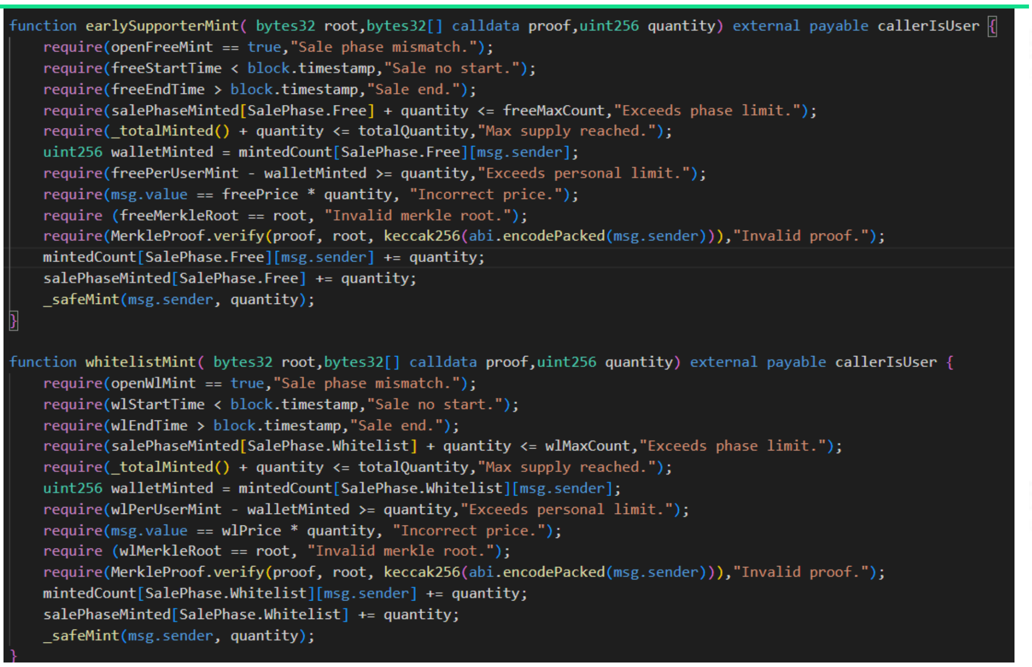
Severity Level	Info
Type	Business Security
Lines	BaddeeBakers.sol #L212,L228
Description	The *earlySupporterMint*, *whitelistMint* and *publicMint* all restrict that the ETH sent by the user should be greater than or equal to the price, but if the ETH sent by the user is greater than the price, the excess funds will be locked in the contract.



```
function earlySupporterMint( bytes32 root,bytes32[] calldata proof,u
    require(openFreeMint == true,"Sale phase mismatch.");
    require(freeStartTime < block.timestamp,"Sale no start.");
    require(freeEndTime > block.timestamp,"Sale end.");
    require(salePhaseMinted[SalePhase.Free] + quantity <= freeMaxCou
    require(_totalMinted() + quantity <= totalQuantity,"Max supply r
    uint256 walletMinted = mintedCount[SalePhase.Free][msg.sender];
    require(freePerUserMint - walletMinted >= quantity,"Exceeds pers
    require(msg.value >= freePrice * quantity, "Incorrect price.");/
    require (freeMerkleRoot == root, "Invalid merkle root.");
    require(MerkleProof.verify(proof, root, keccak256(abi.encodePack
    mintedCount[SalePhase.Free][msg.sender] += quantity;
    salePhaseMinted[SalePhase.Free] += quantity;
    _safeMint(msg.sender, quantity);
}


function whitelistMint( bytes32 root,bytes32[] calldata proof,uint25
    require(openWlMint == true,"Sale phase mismatch.");
    require(wlStartTime < block.timestamp,"Sale no start.");
    require(wlEndTime > block.timestamp,"Sale end.");
    require(salePhaseMinted[SalePhase.Whitelist] + quantity <= wlMax
    require(_totalMinted() + quantity <= totalQuantity,"Max supply r
    uint256 walletMinted = mintedCount[SalePhase.Whitelist][msg.send
    require(wlPerUserMint - walletMinted >= quantity,"Exceeds person
    require(msg.value >= wlPrice * quantity, "Incorrect price.");
    require (wlMerkleRoot == root, "Invalid merkle root.");
    require(MerkleProof.verify(proof, root, keccak256(abi.encodePack
    mintedCount[SalePhase.Whitelist][msg.sender] += quantity;
    salePhaseMinted[SalePhase.Whitelist] += quantity;
    _safeMint(msg.sender, quantity);
```

Figure 5 The source code of *earlySupporterMint, whitelistMint* functions(Unfixed)

Recommendations	1. The code in the red box ">= "changed to "==". 2. Or send excess funds to the user.
Status	Fixed.



```
function earlySupporterMint( bytes32 root,bytes32[] calldata proof,uint256 quantity) external payable callerIsUser {
    require(openFreeMint == true,"Sale phase mismatch.");
    require(freeStartTime < block.timestamp,"Sale no start.");
    require(freeEndTime > block.timestamp,"Sale end.");
    require(salePhaseMinted[SalePhase.Free] + quantity <= freeMaxCount,"Exceeds phase limit.");
    require(_totalMinted() + quantity <= totalQuantity,"Max supply reached.");
    uint256 walletMinted = mintedCount[SalePhase.Free][msg.sender];
    require(freePerUserMint - walletMinted >= quantity,"Exceeds personal limit.");
    require(msg.value == freePrice * quantity, "Incorrect price.");
    require (freeMerkleRoot == root, "Invalid merkle root.");
    require(MerkleProof.verify(proof, root, keccak256(abi.encodePacked(msg.sender))),"Invalid proof.");
    mintedCount[SalePhase.Free][msg.sender] += quantity;
    salePhaseMinted[SalePhase.Free] += quantity;
    _safeMint(msg.sender, quantity);
}

function whitelistMint( bytes32 root,bytes32[] calldata proof,uint256 quantity) external payable callerIsUser {
    require(openWlMint == true,"Sale phase mismatch.");
    require(wlStartTime < block.timestamp,"Sale no start.");
    require(wlEndTime > block.timestamp,"Sale end.");
    require(salePhaseMinted[SalePhase.Whitelist] + quantity <= wlMaxCount,"Exceeds phase limit.");
    require(_totalMinted() + quantity <= totalQuantity,"Max supply reached.");
    uint256 walletMinted = mintedCount[SalePhase.Whitelist][msg.sender];
    require(wlPerUserMint - walletMinted >= quantity,"Exceeds personal limit.");
    require(msg.value == wlPrice * quantity, "Incorrect price.");
    require (wlMerkleRoot == root, "Invalid merkle root.");
    require(MerkleProof.verify(proof, root, keccak256(abi.encodePacked(msg.sender))),"Invalid proof.");
    mintedCount[SalePhase.Whitelist][msg.sender] += quantity;
    salePhaseMinted[SalePhase.Whitelist] += quantity;
    _safeMint(msg.sender, quantity);
}
```

Figure 6 The source code of *earlySupporterMint*, *whitelistMint* functions(Fixed)

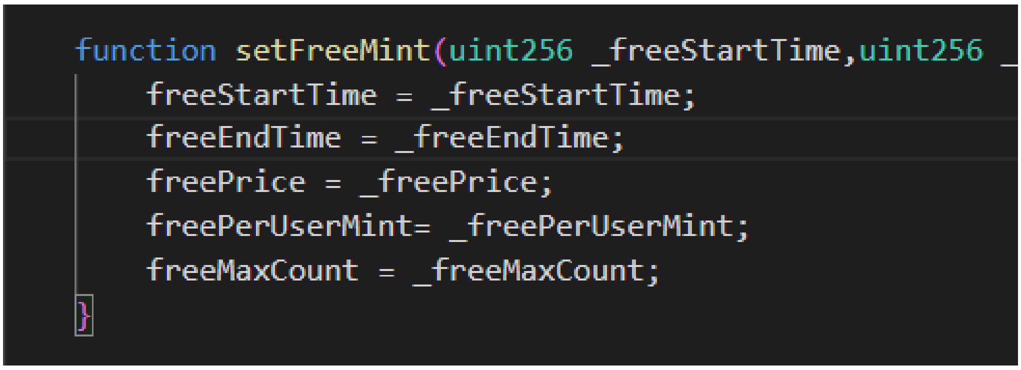
[BaddeeBakers-4] The start time should be less than the end time

Severity Level Info

Type Business Security

Lines BaddeeBakers.sol #L378-L388

Description The *setFreeMint*, *setWhitelistMint* and *setPublicMint* functions can modify the time of the corresponding mint, but do not judge whether the set start time is less than the end time. If the set start time is less than the end time, it will affect the subsequent mint operations.

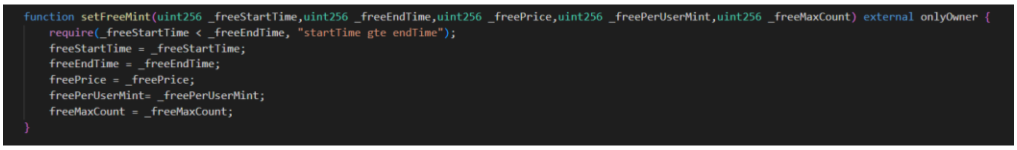


```
function setFreeMint(uint256 _freeStartTime,uint256 _
    freeStartTime = _freeStartTime;
    freeEndTime = _freeEndTime;
    freePrice = _freePrice;
    freePerUserMint= _freePerUserMint;
    freeMaxCount = _freeMaxCount;
}
```

Figure 7 The source code of *setFreeMint* function(Unfixed)

Recommendations It is recommended to added a judgment on the size of start time and end time.

Status Fixed.

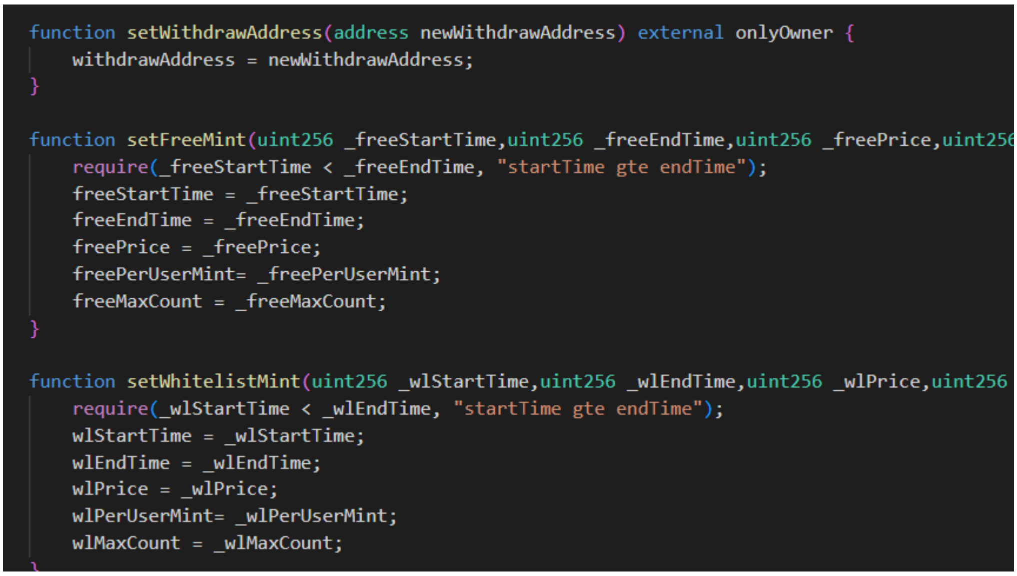


```
function setFreeMint(uint256 _freeStartTime,uint256 _freeEndTime,uint256 _freePrice,uint256 _freePerUserMint,uint256 _freeMaxCount) external onlyOwner {
    require(_freeStartTime < _freeEndTime, "startTime gte endTime");
    freeStartTime = _freeStartTime;
    freeEndTime = _freeEndTime;
    freePrice = _freePrice;
    freePerUserMint= _freePerUserMint;
    freeMaxCount = _freeMaxCount;
}
```

Figure 8 The source code of *setFreeMint* function(Fixed)

[BaddeeBakers-5] The set function missing event

Severity Level	Info
Type	Business Security
Lines	BaddeeBakers.sol #L127,L131,L150,L158
Description	Modifying or setting state variables should add corresponding event notifications.



```
function setWithdrawAddress(address newWithdrawAddress) external onlyOwner {
    withdrawAddress = newWithdrawAddress;
}

function setFreeMint(uint256 _freeStartTime,uint256 _freeEndTime,uint256 _freePrice,uint256
    require(_freeStartTime < _freeEndTime, "startTime gte endTime");
    freeStartTime = _freeStartTime;
    freeEndTime = _freeEndTime;
    freePrice = _freePrice;
    freePerUserMint= _freePerUserMint;
    freeMaxCount = _freeMaxCount;
}

function setWhitelistMint(uint256 _wlStartTime,uint256 _wlEndTime,uint256 _wlPrice,uint256
    require(_wlStartTime < _wlEndTime, "startTime gte endTime");
    wlStartTime = _wlStartTime;
    wlEndTime = _wlEndTime;
    wlPrice = _wlPrice;
    wlPerUserMint= _wlPerUserMint;
    wlMaxCount = _wlMaxCount;
```

Figure 9 The source code of *setWithdrawAddress*, *setFreeMint*, *setWhitelistMint* functions(Unfixed)

Recommendations	It is recommended to add corresponding event notifications.
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 th e 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