

AmazeWallet-nftmarke

t

Smart Contract Security Audit

V1.1

No. 202212131655

Dec 13th, 2022

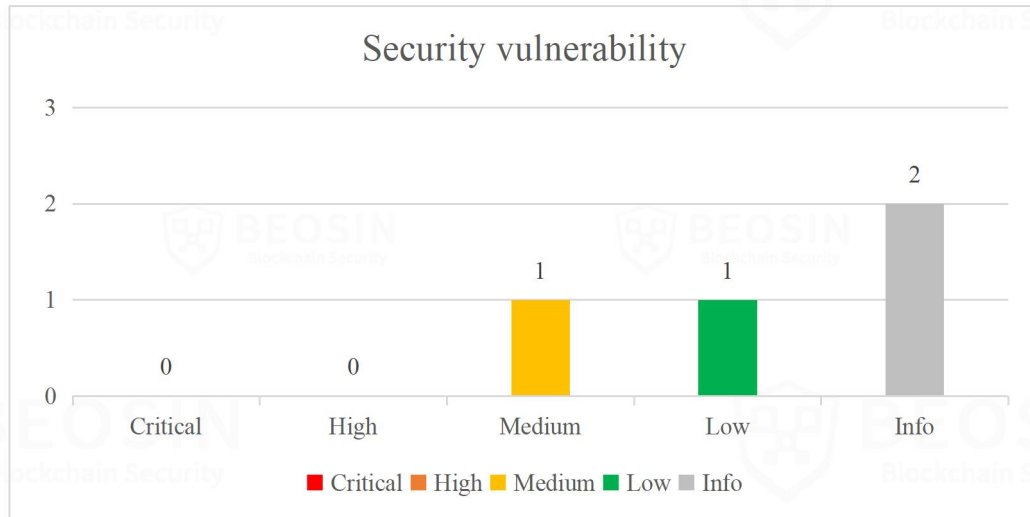


Contents

Summary of Audit Results	1
1 Overview	3
1.1 Project Overview	3
1.2 Audit Overview	3
2 Findings	4
[AmazeWallet-nftmarket-1] Exchangeable roles for both sides of the transaction	5
[AmazeWallet-nftmarket-2] Incomplete signature verification	6
[AmazeWallet-nftmarket-3] <i>mint</i> function have no permission restrictions	7
[AmazeWallet-nftmarket-4] No trigger event	8
3 Appendix	9
3.1 Vulnerability Assessment Metrics and Status in Smart Contracts	9
3.2 Audit Categories	11
3.3 Disclaimer	13
3.4 About BEOSIN	14

Summary of Audit Results

After auditing, 1 Medium, 1 Low and 2 Info-risk items was identified in the AmazeWallet-nftmarket 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. The *exchangeFixedPrice* function does not judge the signature Nonce, the same ERC1155 order can be purchased multiple times.
2. The *mint* function in AmazeWalletTree contract has no permission restrictions, anyone can call it for casting.

- **Project Description:**

- 1. Business overview**

AmazeWallet-nftmarket is an NFT trading platform that supports both ERC1155 and ERC721 standards. Users can only use the whitelisted NFT in the contract and the specified payment ERC20 tokens for transactions. During the transaction, the caller needs to provide the transaction signatures of both buyers and sellers, and both buyers and sellers are required to authorize the contract in advance. The commission ranges from 0 to 10% and can be set by the seller. In the MTCNFT721 and MTCNFT1155 contracts, only the *safeTransferFrom* function, which can be called by the operator, will mint the ERC721 or ERC1155 tokens with the specified id before sending them if they do not exist.

1 Overview

1.1 Project Overview

Project Name	AmazeWallet-nftmarket
Platform	BNB Chain
File Hash(SHA-256)	39ed3db8b428148c48a3aa25dcded0d3b64f681d1bde91fa1f5b90896a1032bf (Unfixed) 9a78ca96effd9a6a029b28948a5fb1f30fcf1fd18d6cefbcb68fb2eac25e017b2 (Fixed)

1.2 Audit Overview

Audit work duration: Dec 7, 2022 – Dec 13, 2022

Update date: Dec 16

Update details: Add problem fix instructions

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
AmazeWallet-nftmarket-1	Exchangeable roles for both sides of the transaction	Medium	Fixed
AmazeWallet-nftmarket-2	Incomplete signature verification	Low	Fixed
AmazeWallet-nftmarket-3	<i>mint</i> function have no permission restrictions	Info	Acknowledged
AmazeWallet-nftmarket-4	No trigger event	Info	Acknowledged

Status Notes:

1. AmazeWallet-nftmarket-3 is not fixed and may not cause any issue.
2. AmazeWallet-nftmarket-4 is nor fixed and may not cause any issue.

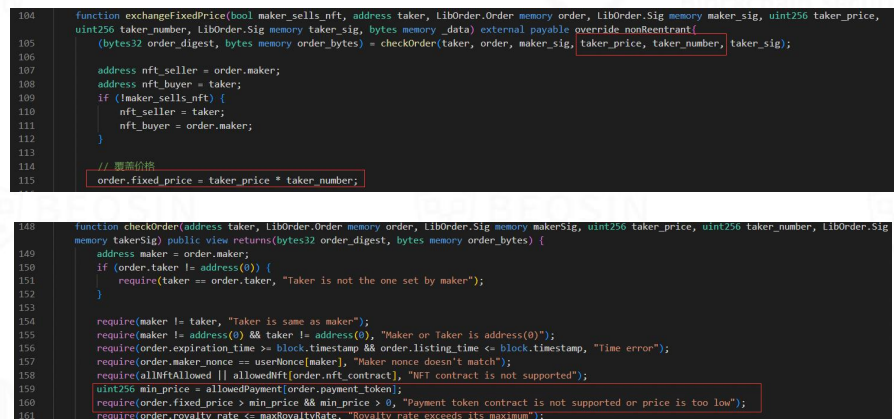
Finding Details:

[AmazeWallet-nftmarket-1] Exchangeable roles for both sides of the transaction

Severity Level	Medium
Type	Business Security
Lines	AmazeWallet-nftmarketV2.sol#L104-112
Description	<p>A bool value that is unchecked and entered by the caller can reverse the roles of the buyer and seller and may result maker change from a seller to a buyer, this may cause the two parties to exchange, and the executed transaction content does not match the signature. The seller may lose tokens.</p> <pre> 104 function exchangeFixedPrice(bool maker_sells_nft, address taker, LibOrder.Order memory order, LibOrder.Sig memory maker_sig, uint256 taker_price, 105 uint256 taker_number, LibOrder.Sig memory taker_sig, bytes memory _data) external payable override nonReentrant{ 106 (bytes32 order_digest, bytes memory order_bytes) = checkOrder(taker, order, maker_sig, taker_price, taker_number, taker_sig); 107 108 address nft_seller = order.maker; 109 address nft_buyer = taker; 110 if (!maker_sells_nft) { 111 nft_seller = taker; 112 nft_buyer = order.maker; 113 } </pre>
Recommendations	It is recommended to specify in the maker's manufacturing order whether the maker is a nft seller.
Status	Fixed. It has been deleted.

[AmazeWallet-nftmarket-2] Incomplete signature verification

Severity Level	Low
Type	Business Security
Lines	AmazeWallet-nftmarketV2.sol#L104-115 AmazeWallet-nftmarketV2.sol#L148-161
Description	The AmazeWallet-nftmarketV2 contract <i>exchangeFixedPrice</i> in the order completion, the signature is judged only on the unit price and quantity of this transaction. In order to meet the purchase demand of ERC1155, nonce is not used in the signature. This may cause the same ERC1155 order can be purchased multiple times, and only the total price is checked in the <i>checkOrder</i> function, not the unit price and quantity, subsequent purchase taker may pass in the wrong data. When the sold nft is returned to the seller's account, the nft can still be sold by the previous order (If the order expiration time is not reached).



```

104 function exchangeFixedPrice(bool maker_sells_nft, address taker, LibOrder.Order memory order, LibOrder.Sig memory maker_sig, uint256 taker_price,
105 uint256 taker_number, LibOrder.Sig memory taker_sig, bytes memory_data) external payable override nonReentrant(
106 (bytes32 order_digest, bytes memory order_bytes) = checkOrder(taker, order, maker_sig, taker_price, taker_number, taker_sig);
107
108 address nft_seller = order.maker;
109 address nft_buyer = taker;
110 if (!maker_sells_nft) {
111     nft_seller = taker;
112     nft_buyer = order.maker;
113 }
114 // 需要的话
115 order.fixed_price = taker_price * taker_number;

```

```

148 function checkOrder(address taker, LibOrder.Order memory order, LibOrder.Sig memory makersig, uint256 taker_price, uint256 taker_number, LibOrder.Sig
149 memory takersig) public view returns(bytes32 order_digest, bytes memory order_bytes) {
150     address maker = order.maker;
151     if (order.taker != address(0)) {
152         require(taker == order.taker, "Taker is not the one set by maker");
153     }
154     require(maker != taker, "Taker is same as maker");
155     require(maker != address(0) && taker != address(0), "Maker or Taker is address(0)");
156     require(order.expiration_time >= block.timestamp && order.listing_time <= block.timestamp, "Time error");
157     require(order.maker_nonce == userNonce[maker], "Maker nonce doesn't match");
158     require(!lib1155allowed || allowedNft(order.nft_contract), "NFT contract is not supported");
159     uint256 min_price = allowedPayment[order.payment_token];
160     require(order.fixed_price > min_price && min_price > 0, "Payment token contract is not supported or price is too low");
161     require(order.royalty_rate <= maxRoyaltyRate, "Royalty rate exceeds its maximum");

```

Figure 1 Source code of related functions (Unfixed)

Recommendations It is recommended to choose whether to add the nonce mechanism according to the actual business situation, increase the judgment of unit price and quantity, or prompt the user to cancel the authorization of this contract after the order is filled.

Status Fixed. Added restrictions on unit price and quantity.



```

156 require(order.fixed_price > min_price && min_price > 0, "Payment token contract is not supported or price is too low");
157 require(taker_number > 0, "Taker number error");
158 if (order.amount > 0) {
159     require(taker_number <= order.amount, "Taker number error");
160 }
161 require(taker_price >= order.fixed_price, "Taker price error");
162 require(order.royalty_rate <= maxRoyaltyRate, "Royalty rate exceeds its maximum");

```

Figure 2 Source code of related functions (Fixed)

[AmazeWallet-nftmarket-3] *mint* function have no permission restrictions

Severity Level	Info
Type	Business Security
Lines	AmazeWalletTree.sol#L41-43
Description	<p>When the contract is in an unpaused state, anyone can mint NFT tokens for themselves through the <i>mint</i> function, and the paused state only affects the <i>mint</i> function, not functions such as <i>transferFrom</i>.</p> <pre> 41 function mint() external override whenNotPaused { 42 _safeMint(_msgSender(), totalSupply++); 43 }</pre>
Recommendations	It is recommended to restricted to owner calls only.
Status	Acknowledged.

Figure 3 Source code of *mint* function

[AmazeWallet-nftmarket-4] No trigger event

Severity Level	Info
Type	Business Security
Lines	AmazeWallet-nftmarketV2.sol#L61-102
Description	Owner modified keyword parameters did not touch the event.

```

61 function setOperator(address addr, bool flag) external onlyOwner {
62     if (!flag) {
63         delete operators[addr];
64     } else {
65         operators[addr] = true;
66     }
67 }
68
69 function setFeeRate(uint256 _feeRate) external onlyOwner {
70     require(_feeRate <= feeDenominator, "Fee rate is too high");
71     feeRate = _feeRate;
72 }
73
74 function setFeeRecipient(address _feeRecipient) external onlyOwner {
75     require(_feeRecipient != address(0), "Fee recipient rate is address(0)");
76     feeRecipient = _feeRecipient;
77 }
78
79 function setMaxRoyaltyRate(uint256 _maxRoyaltyRate) external onlyOwner {
80     require(_maxRoyaltyRate <= feeDenominator, "Royalty rate is too high");
81     maxRoyaltyRate = _maxRoyaltyRate;
82 }
83
84 function setAllNftAllowed(bool _allNftAllowed) external onlyOperator {
85     allNftAllowed = _allNftAllowed;
86 }
87
88 function setNftAllowed(address _contract, bool flag) external onlyOperator {
89     if (!flag) {
90         delete allowedNft[_contract];
91     } else {
92         allowedNft[_contract] = true;
93     }
94 }
95
96 function setPaymentAllowed(address _contract, uint256 min_price) external onlyOperator {
97     if (min_price == 0) {
98         delete allowedPayment[_contract];
99     } else {
100         allowedPayment[_contract] = min_price;
101     }
102 }

```

Figure 4 Source code of related functions

Recommendations	It is recommended to add and trigger corresponding events.
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.



Official Website

<https://www.beosin.com>

Telegram

<https://t.me/+dD8Bnqd133RmNWNl>

Twitter

https://twitter.com/Beosin_com

Email

Contact@beosin.com

