BEOSIN
Blockchain Security

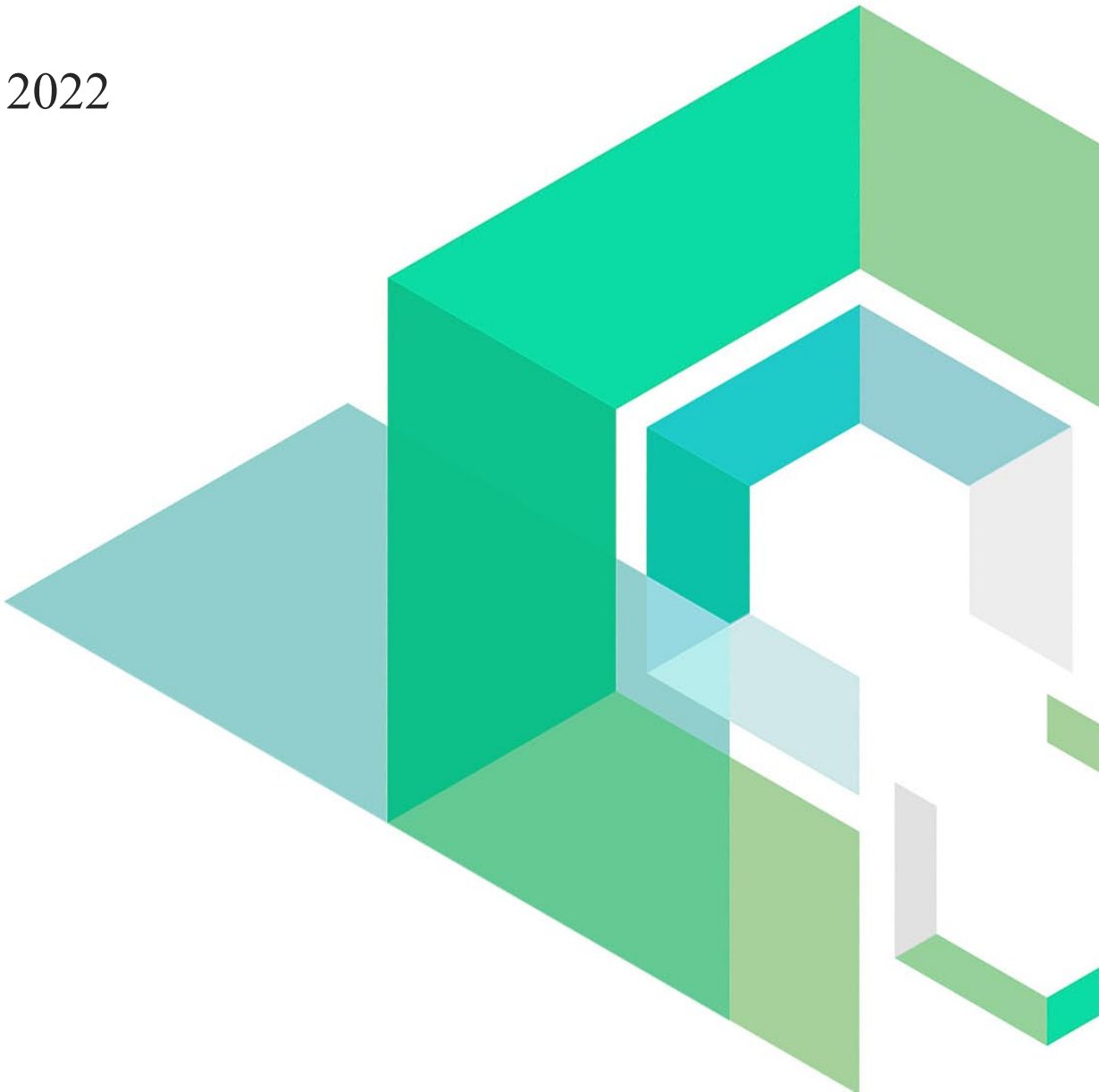
IvyMarket

Smart Contract Security Audit

V1.0

No. 202209231629

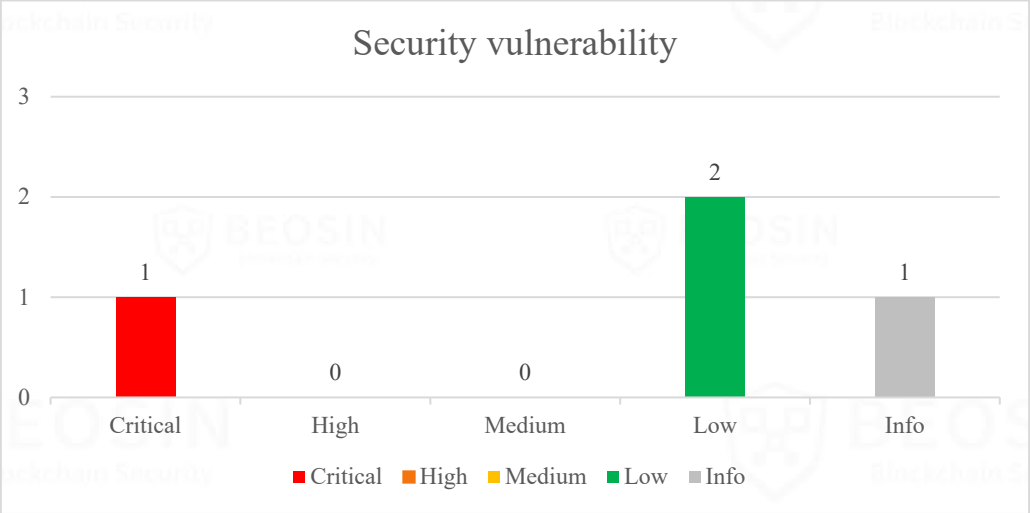
Sep 23th, 2022



Contents

Summary of Audit Results

After auditing, 1 Critical-risk, 2 Low-risk and 1 Info items were identified in the IvyMarket 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：**

  The proxy contract undertakes the authorization of all tokens interacting with the Exchange contract. In addition, the proxy contract management address is now specified by the project party and is not managed by multi signature addresses. Users should pay attention to the security of the private key of the management address in real time.

- **Notes for users:**

1. For the IvyAsset contract, the security of the three-party address should be taken care when the user uses *setApproveForAll* for authorization. otherwise the three-party address will likely cast and transfer all the NFTS belonging to the user. In addition, if this contract interacts with other projects, use the *safeTransferFrom* function to avoid asset loss caused by reentry through the *onNRC721Received* function of the target contract.
2. When placing a manufacturing order, the user should pay attention to setting the end time of the order reasonably. Otherwise, if the low-price order is forgotten to be cancelled, it may be sold at a low price when the NFT price rises.

- **Project Description:**

This audit includes the IvyMarket contract, the Proxy contract and the IvyAssert NFT contract.

The IvyMarket contract is used for matching of buy and sell pending orders. It will call the Proxy contract related functions for asset transfer processing; the Proxy contract is used to store user asset authorization, and the contract can be used as the user's agent to transfer tokens; the IvyAssert NFT contract is NRC721 standard NFT, in the IvyAssert NFT contract every time Each user address has its own unique tokenId prefix, and NFT tokens with the same tokenId prefix belong to the same address by default. (For example: the prefix of an address in IA NFT is 1480746774692272938771265728976432807095523939003056004232684946639190549985535470574209469 6, then the NFTs identified by tokenIDs such as “1480…94696001”, “1480…94696002”, etc. initially belong to this address)

1 Overview

1.1 Project Overview

Project Name	IvyMarket
Platform	NULS
Audit scope 1	https://github.com/IvyMarket/NULS-NRC721-baselib
Commit Hash 1	d66b0fc393d19992194a42e3cb80851eaf997d0b
Audit scope 2	https://github.com/IvyMarket/nuls-contracts
Audit scope 2	f18f76dd584f7dd2e2fdbc276b233c63229f3e8b (Unfixed) 07c39eb5662e8f30d76619887c69757744820edc (Fixed)

1.2 Audit Overview

Audit work duration: September 10, 2022 – September 23, 2022

Audit functions: Formal Verification, Static Analysis, Typical Case Testing and Manual Review.

Audit team: Beosin Security Team

2 Findings

Index	Risk description	Severity level	Status
IVY-1	The order.calldata lacks parameter dependency validation	Critical	Fixed
IVY-2	Centralization risk	Low	Acknowledged
IVY-3	The newMinimumMakerProtocolFee and minimumTakerProtocolFee are missing cap limit	Low	Fixed
IVY-4	Redundant code	Info	Fixed

Status Notes:

1. IVY-2 is acknowledged by the IvyMarket project party. If the private key of the owner account of the contract is stolen, all authorized tokens will be at risk of being stolen.

Finding Details:

[IVY-1] The order.calldata lacks parameter dependency validation

Severity Level	**Critical**
Type	Business Security
Lines	Ivy-contracts/src/main/java/io/ivymarket/exchange/ExchangeCore.java #L439-450
Description	Since the from address in order.calldata lacks the dependency-check with the parameters of the order, it is possible to construct malicious calldata in the buy and sell orders to transfer all the NFTs authorized to the proxy contract.

```
protected boolean calldataCanMatch(Calldata buy, Calldata sell) {
    if (!((!ZERO_ADDR.equals(buy.getFrom()) && ZERO_ADDR.equals(buy.getTo()) && ZERO_ADDR.equals(sell.getFrom()) && !ZERO_ADDR.equals(sell.getTo()))
            || (ZERO_ADDR.equals(buy.getFrom()) && !ZERO_ADDR.equals(buy.getTo()) && !ZERO_ADDR.equals(sell.getFrom()) && ZERO_ADDR.equals(sell.getTo())))) {
        return false;
    }

    if (buy.getTokenId().compareTo(sell.getTokenId()) != 0) {
        return false;
    }

    return buy.getData().equals(sell.getData());
}
```

Figure 1 *calldataCanMatch* function (unfixed)

Recommendations	It is recommended to check the consistency of order parameters and order.calldata.
Status	Fixed.

```
protected boolean calldataCanMatch(Calldata buy, Address buyMaker, Calldata sell, Address sellMaker) {
    if (!(ZERO_ADDR.equals(buy.getFrom()) && !ZERO_ADDR.equals(buy.getTo()) &&
            !ZERO_ADDR.equals(sell.getFrom()) && ZERO_ADDR.equals(sell.getTo()) &&
            buy.getTo().equals(buyMaker) && sell.getFrom().equals(sellMaker))) {
        return false;
    }

    if (buy.getTokenId().compareTo(sell.getTokenId()) != 0) {
        return false;
    }

    return buy.getData().equals(sell.getData());
}
```

Figure 2 *calldataCanMatch* function (fixed)

[IVY-2] Centralization risk

Severity Level	**Low**
Type	Business Security
Lines	proxy-contracts/src/main/java/io/ivymarket/contract/ProxyContract.java
Description	Tokens that interact with the Exchange contract need to be authorized to the ProxyContact contract, and the *transferERC20*, *transferNFT721* and *transferNFT1155* functions are used for proxy transfer in the ProxyContact. The above functions only check that the caller is an authorized address, and there can be multiple authorized addresses. If the private key of the authorized address is stolen, all authorized tokens will be at risk of being stolen.

```
public Boolean transferERC20(@Required Address target, @Required Address from, @Required Address to, @Required BigInteger value) {
    require(!pause, Errors.ERROR_TRANSFER_PAUSED);
    Boolean auth = contracts.get(Msg.sender());
    require(auth != null && auth, Errors.ERROR_TRANSFER_NRC20);
    String[][] args = new String[][]{
            new String[]{from.toString()},
            new String[]{to.toString()},
            new String[]{value.toString()}
    };
    String returnValue = target.callWithReturnValue("transferFrom", null, args, BigInteger.ZERO);
    return Boolean.valueOf(returnValue);
}
```

Figure 3 *transferERC20* function

```
public void transferNFT721(@Required Address target, @Required Address from, @Required Address to, @Required BigInteger tokenId, String data) {
    require(!pause, Errors.ERROR_TRANSFER_PAUSED);
    Boolean auth = contracts.get(Msg.sender());
    require(auth != null && auth, Errors.ERROR_TRANSFER_NRC721);
    if (data == null || data.trim().isEmpty()) {
        String[][] args = new String[][]{
                new String[]{from.toString()},
                new String[]{to.toString()},
                new String[]{tokenId.toString()},
        };
        target.call("safeTransferFrom", "(Address from, Address to, BigInteger tokenId) return void", args, BigInteger.ZERO);
    } else {
        String[][] args = new String[][]{
                new String[]{from.toString()},
                new String[]{to.toString()},
                new String[]{tokenId.toString()},
                new String[]{data}
        };
        target.call("safeTransferFrom", "(Address from, Address to, BigInteger tokenId, String data) return void", args, BigInteger.ZERO);
    }
}
```

Figure 4 *transferNFT721* function

```
public void transferNFT1155(@Required Address target, @Required Address from, @Required Address to, @Required BigInteger tokenId, @Required BigInteger amount, String data) {
    require(!pause, Errors.ERROR_TRANSFER_PAUSED);
    Boolean auth = contracts.get(Msg.sender());
    require(auth != null && auth, Errors.ERROR_TRANSFER_NRC1155);
    if (data == null || data.trim().isEmpty()) {
        String[][] args = new String[][]{
                new String[]{from.toString()},
                new String[]{to.toString()},
                new String[]{tokenId.toString()},
                new String[]{amount.toString()}
        };
        target.call("safeTransferFrom", "(Address from, Address to, BigInteger tokenId, BigInteger amount) return void", args, BigInteger.ZERO);
    } else {
        String[][] args = new String[][]{
                new String[]{from.toString()},
                new String[]{to.toString()},
                new String[]{tokenId.toString()},
                new String[]{amount.toString()},
                new String[]{data}};
        target.call("safeTransferFrom", "(Address from, Address to, BigInteger tokenId, BigInteger amount, String data) return void", args, BigInteger.ZERO);
    }
}
```

Figure 5 *transferNFT721* function

Recommendations	It is recommended to use a variable instead of mapping to store Exchange contract. We also recommend to use the multi-signature address to manage the owner of the ProxyContract.
Status	Acknowledged. According to the project party, mapping is considered in the code that the exchange contract may be upgraded, which is compatible with the previous version of order, and supports multi-signature management in the future.

[IVY-3] The newMinimumMakerProtocolFee and minimumTakerProtocolFee are missing cap limit

Severity Level	**Low**
Type	Business Security
Lines	Ivy-contracts/src/main/java/io/ivymarket/exchange/ExchangeCore.java #L75-78 Ivy-contracts/src/main/java/io/ivymarket/exchange/ExchangeCore.java #L92-95
Description	The newMinimumMakerProtocolFee and NewMinimumTakerProtocolFee without upper bound limitation, which may cause transactions failure when the fee exceeds 10,000.

```
public void changeMinimumMakerProtocolFee(BigInteger newMinimumMakerProtocolFee) {
    onlyOwner();
    minimumMakerProtocolFee = newMinimumMakerProtocolFee;
}
```

Figure 6 *changeMinimumMakerProtocolFee* function (unfixed)

```
public void changeMinimumTakerProtocolFee(BigInteger newMinimumTakerProtocolFee) {
    onlyOwner();
    minimumTakerProtocolFee = newMinimumTakerProtocolFee;
}
```

Figure 7 *changeMinimumTakerProtocolFee* function (unfixed)

Recommendations	It is recommended to add a valid range for preset values in the related functions which can set newMinimumMakerProtocolFee and newMinimumTakerProtocolFee to avoid excessive fees.
Status	Fixed.

```
public void changeMinimumMakerProtocolFee(BigInteger newMinimumMakerProtocolFee) {
    onlyOwner();
    require(newMinimumMakerProtocolFee.compareTo(INVERSE_BASIS_POINT) < 0, ERROR__MIN_MAKER_PROTOCOL_FEE);
    minimumMakerProtocolFee = newMinimumMakerProtocolFee;
}
```

Figure 8 *changeMinimumMakerProtocolFee* function (fixed)

```
public void changeMinimumTakerProtocolFee(BigInteger newMinimumTakerProtocolFee) {
    onlyOwner();
    require(newMinimumTakerProtocolFee.compareTo(INVERSE_BASIS_POINT) < 0, ERROR_MIN_TAKER_PROTOCOL_FEE);
    minimumTakerProtocolFee = newMinimumTakerProtocolFee;
}
```

Figure 9 *changeMinimumTakerProtocolFee* function (unfixed)

[IVY-4] Redundant code

Severity Level	Info
Type	Coding Conventions
Lines	proxy-contracts/src/main/java/io/ivymarket/contract/ProxyContract.java #L3
Description	Unused package is introduced in ProxyContract.

```
import io.netty.util.internal.StringUtil;
import io.nuls.contract.sdk.Address;
import io.nuls.contract.sdk.Contract;
import io.nuls.contract.sdk.Msg;
import io.nuls.contract.sdk.annotation.Required;
import io.nuls.contract.sdk.annotation.View;
```

Figure 10 Redundant code

Recommendations	It is recommended to remove the redundant code.
Status	Fixed.

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