BEOSIN
Blockchain Security

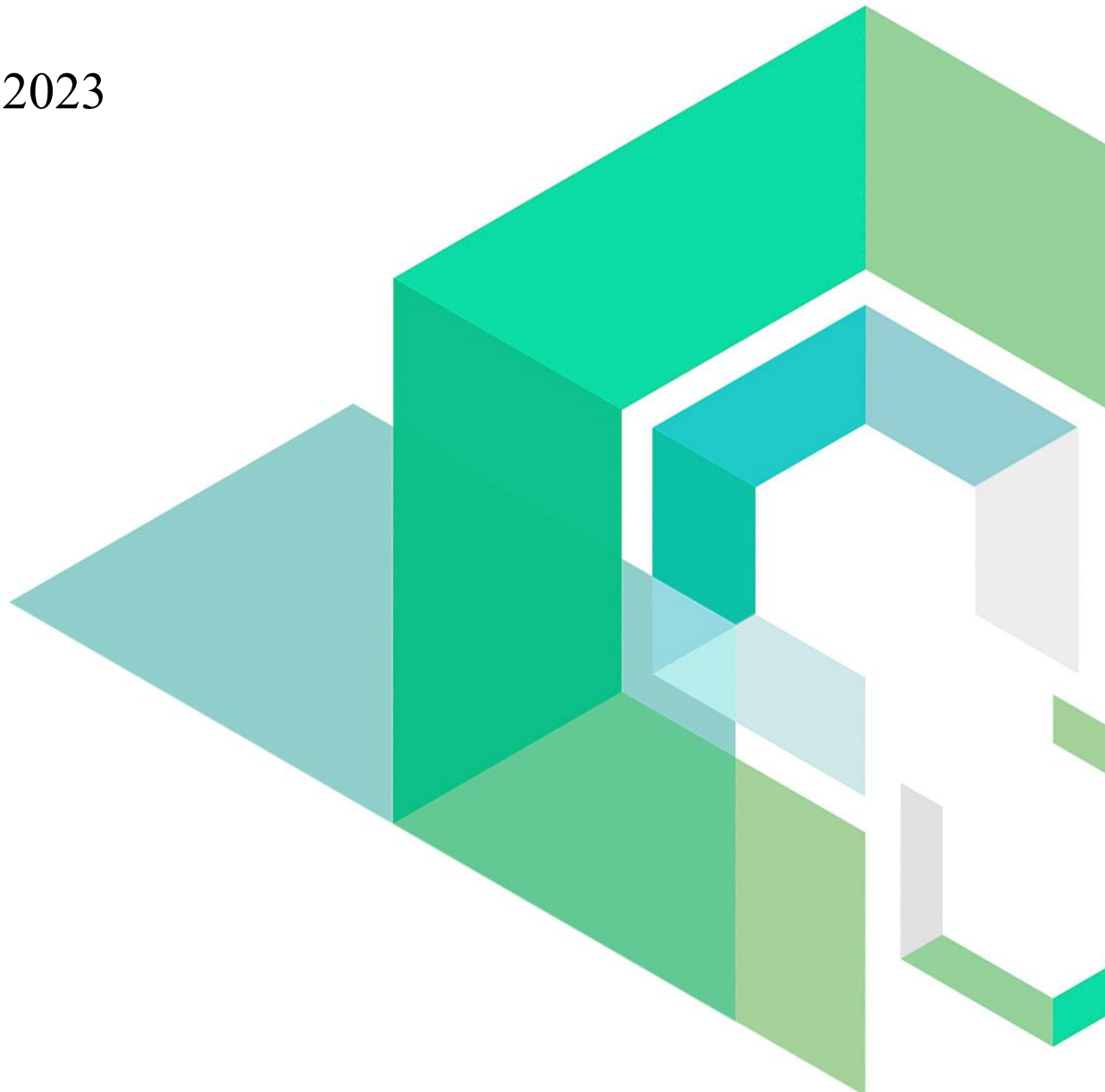
Dfans

Smart Contract Security Audit

V1.0

No. 202305081114

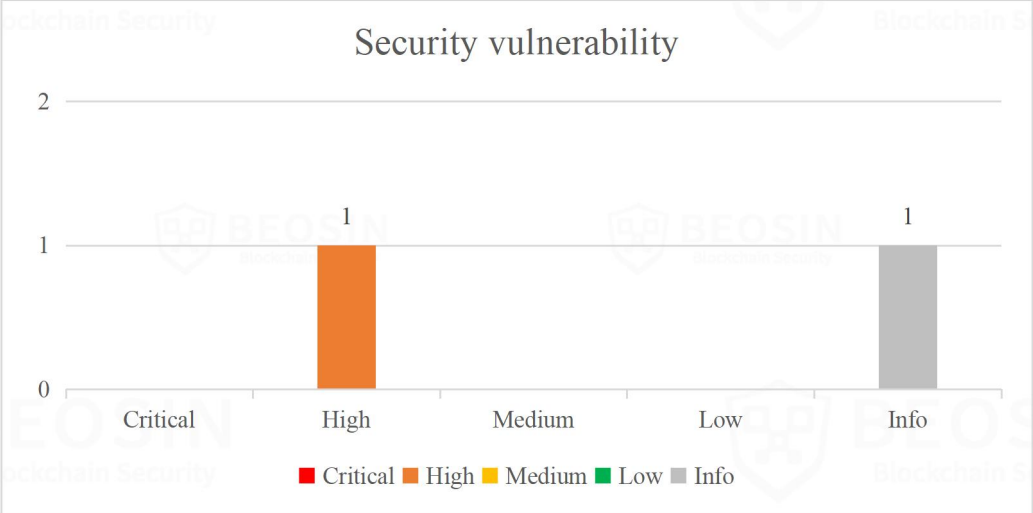
May 8th, 2023



Contents

Summary of Audit Results

After auditing, 1 High-risk and 1 Info item were identified in the Dfans project. Specific audit details will be presented in the **Findings** section. Users should pay attention to the following aspects when interacting with this project:



All the problem items found in this project have been fixed, but the execution of some functions in the contract depends on the signature data of the project signers. When using it, please ensure that the signature data is correct.

- **Project Description:**

Business overview

Dfans is a project for users to issue NFT contracts, mainly including two main contracts, DfansPassCreator and DfansPass. The function of the DfansPassCreator contract is to deploy the NFT contract with DfansPass as the template. When deploying, the user can specify parameters, such as NFT name, symbol, initialPublish, etc., but the administrator owner is always the owner account of the DfansPassCreator contract.

The DfansPass contract is based on ERC721, adding the functions of NFT issuance, pause and dividends for NFT holders.

- NFT issuance: The contract has 4 ways to issue NFT, *safeMint*, *safeBatchMint*, *publicMint* and *publicRandomMint*. Among them, *safeMint* and *safeBatchMint* can only be called by the owner to mint NFT for the specified address; *publicMint* and *publicRandomMint* will be called by the user to choose to purchase NFT at a fixed price according to the fixedPrice configured in the contract, or purchase according to the signature of the signers.
- Pause: The owner of the contract can pause the contract, after the pause, all users will no longer be able to mint, transfer the NFTs, and cannot withdraw rewards; of course, the owner can also lift the pause of the contract.
- Dividends for holding NFT tokens: DfansPass contract will accept payment from other accounts, and the contract administrator owner can call the release function and distribute this part of assets to NFT holders.

1 Overview

1.1 Project Overview

Project Name	Dfans
Platform	Ethereum
Project Link	https://gitee.com/Worktogether-Singapore/dfans-contract/tree/master
Commit Hash	3b204d8d69a03a5c03c6ce436043a62294060851 61bf2963818f097f54bb6e87e39b1c044fd02862 e1bfecc12d0e5df6b51a0f086c42bc9a0d6d1e52 02c18dd3fc0ec4cad26ed791c0b7950d50c4f33e

1.2 Audit Overview

Audit work duration: May 5, 2023 – May 8, 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
Dfans-1	Missing checks for NFT purchase amount	High	Fixed
Dfans-2	The withdraw function is not affected by pause	Info	Fixed

Finding Details:

[Dfans-1] Missing checks for NFT purchase amount

Severity Level	High
Type	Business Security
Lines	DfansPass.sol#L162-225
Description	As shown in the figure below, when users purchase NFT through the *publicRandomMint* function, they can enter the parameter n to indicate the quantity of this purchase, but the fee to be paid is only the price of one NFT.

```
162    function publicRandomMint(
163        address to,
164        uint256 price,
165        string calldata nonce,
166        bytes calldata sig,
167        uint256 n
168    ) public payable callerIsUser {
169        /**
170         * ##### CHECKS
171         */
172
173        // value is enough
174        require(msg.value >= price, "mint value is not enough");
175
176        // if fixedPrice is not 0, check price must bigger than that
177        require(fixedPrice == 0 || price >= fixedPrice , "mint value is not enough");
178
179        // simple check
180        require(_mintPosition.current() < _totalPublished.current(), "token sold out");
181        // check there are enough nfts
182        // find next n tokenid to mint
183        uint256[] memory tokens = new uint256[](n);
184        uint256 filled = 0;
185        uint256 position = _mintPosition.current();
186        while (position < _totalPublished.current() && filled < n) {
187            if (_ownerOf(position) == address(0)) {
188                // find one fit
189                tokens[filled] = position;
```

Figure 1 Source code of *publicRandomMint* function

Recommendations	The fix of this problem needs to consider two aspects. When using a fixed price to purchase, the amount of ETH paid by the user should be price * n; when using a signature to purchase, you need to consider whether the price in the signature has already calculated n NFTs total price, if it is included, there is no need to make changes, if it is not included, you also need to use price * n to check the user's payment amount.
Status	Fixed. For fixedPrice, the function is modified to use n * fixedPrice to calculate the ETH that users need to pay; for selling NFTs using signatures, the project party said that it will calculate the price in advance based on the user's purchase quantity, and then sign.

```
function publicRandomMint(
    address to,
    uint256 price,
    string calldata nonce,
    bytes calldata sig,
    uint256 n
) public payable callerIsUser {
    /**
     * ##### CHECKS
     */

    // value is enough
    require(msg.value >= price, "mint value is not enough");

    // if fixedPrice is not 0, check price must bigger than that
    require(fixedPrice == 0 || price >= n * fixedPrice , "mint value is not enough");

    // simple check
    require(_mintPosition.current() < _totalPublished.current(), "token sold out");
    // check there are enough nfts
    // find next n tokenid to mint
    uint256[] memory tokens = new uint256[](n);
    uint256 filled = 0;
    uint256 position = _mintPosition.current();
    while (position < _totalPublished.current() && filled < n) {
        if (_ownerOf(position) == address(0)) {
            // find one fit
            tokens[filled] = position;
            filled++;
        }
        position++;
    }
```

Figure 2 Source code of *publicRandomMint* function (Fixed)

[Dfans-2] The withdraw function is not affected by pause

Severity Level	Info
Type	Business Security
Lines	DfansPass.sol#L389-403
Description	The pause module is implemented in the contract. Currently, the pause module is used for NFT transfers and mint, as well as the *safePublish* function of the main contract, while the withdraw function is not restricted by the pause module.

```
function withdraw() external {

    uint256 payment = withdrawable(_msgSender());
    require(payment != 0, "account is not due payment");

    _withdrawn[_msgSender()] += payment;
    _transfer(_msgSender(), payment);
    emit PaymentWithdrawn(_msgSender(), payment);
}

function withdrawForBeneficiary() external {
    uint256 withdrawableForBeneficiary = payable(address(this)).balance - totalReceived;
    require(withdrawableForBeneficiary > 0, "nothing to withdraw");
    _transfer(beneficiary, withdrawableForBeneficiary);
}
```

Figure 3 Source code of *withdraw* and *withdrawForBeneficiary* functions

Recommendations	It is recommended to apply the pause module to the withdraw function. When the contract is paused, the user cannot withdraw rewards.
Status	Fixed. In the new version of the code, the *withdraw* and *withdrawForBeneficiary* functions have been restricted by the whenNotPaused modifier. After the contract is suspended, these two functions will not be able to be called

```
function withdraw() external whenNotPaused {

    uint256 payment = withdrawable(_msgSender());
    require(payment != 0, "account is not due payment");

    _withdrawn[_msgSender()] += payment;
    _transfer(_msgSender(), payment);
    emit PaymentWithdrawn(_msgSender(), payment);
}

function withdrawForBeneficiary() external whenNotPaused {
    uint256 withdrawableForBeneficiary = payable(address(this)).balance - totalReceived;
    require(withdrawableForBeneficiary > 0, "nothing to withdraw");
    _transfer(beneficiary, withdrawableForBeneficiary);
}
```

Figure 4 Source code of *withdraw* and *withdrawForBeneficiary* functions (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	Redundant Code
		require/assert Usage
		Cycles Consumption
2	General Vulnerability	Integer Overflow/Underflow
		Reentrancy
		Pseudo-random Number Generator (PRNG)
		Transaction-Ordering Dependence
		DoS (Denial of Service)
		Function Call Permissions
		Returned Value Security
		Rollback Risk
		Replay Attack
		Overriding Variables
		Call Canister controllable
		Canister upgrade risk
		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