
# Security Audit Report for Paras Token Contract

**Date:** Aug 30, 2022

**Version:** 1.0

**Contact**: contact@blocksec.com

BlockSec Audit Team

Digitally signed by BlockSec Audit Team
Date: 2022.08.30 17:09:00 +08'00'

# Contents


**1 Introduction** **1**

1.1 About Target Contracts . . . 1
1.2 Disclaimer . . . 1
1.3 Procedure of Auditing . . . 1
1.3.1 Software Security . . . 2
1.3.2 DeFi Security . . . 2
1.3.3 NFT Security . . . 2
1.3.4 Additional Recommendation . . . 2
1.4 Security Model . . . 3

**2 Findings** **4**

2.1 Additional Recommendation . . . 4
2.1.1 Inconsistency of Constant Name . . . 4
2.2 Notes . . . 4
2.2.1 Assumption on the Secure Implementation of NEAR SDK . . . 4
2.2.2 Fixed Total Supply Owned by the Contract Owner . . . 5

## Report Manifest

| Item | Description |
| --- | --- |
| Client | Paras |
| Target | Paras Token Contract |

## Version History

| Version | Date | Description |
| --- | --- | --- |
| 1.0 | Aug 30, 2022 | First Release |

**About BlockSec** The BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 5 million dollars by blocking multiple attacks. They can be reached at Email, Twitter and Medium.

# Chapter 1 Introduction

## 1.1 About Target Contracts

| Information | Description |
|---|---|
| Type | Smart Contract |
| Language | Rust |
| Approach | Semi-automatic and manual verification |

The repository that has been audited includes the **PARAS** token contract [1].

The auditing process is iterative. Specifically, we will audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following. Our audit report is responsible for the only initial version (`Version 1` ), as well as new codes (in the following versions) to fix issues in the audit report.

| Project | | Commit SHA |
|---|---|---|
| Paras Token Contract | `Version 1` | `8b7ffcbd2bc00bb08c8898096ec09f8c9ef88ef3` |

Note that, we did **NOT** audit all the modules in the repository. The modules covered by this audit report include **ft** folder contract only. Specifically, the file covered in this audit include:

- src/lib.rs

## 1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

## 1.3 Procedure of Auditing

We perform the audit according to the following procedure.

[1] https://github.com/ParasHQ/paras-token-contract

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

### 1.3.1 Software Security

* Reentrancy
* DoS
* Access control
* Data handling and data flow
* Exception handling
* Untrusted external call and control flow
* Initialization consistency
* Events operation
* Error-prone randomness
* Improper use of the proxy system

### 1.3.2 DeFi Security

* Semantic consistency
* Functionality consistency
* Permission management
* Business logic
* Token operation
* Emergency mechanism
* Oracle security
* Whitelist and blacklist
* Economic impact
* Batch transfer

### 1.3.3 NFT Security

* Duplicated item
* Verification of the token receiver
* Off-chain metadata security

### 1.3.4 Additional Recommendation

* Gas optimization

* Code quality and style

**Note** *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

## 1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [2] and Common Weakness Enumeration [3]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

**Table 1.1:** Vulnerability Severity Classification

| Impact \ Likelihood | *High* | *Low* |
|---|---|---|
| *High* | High | Medium |
| *Low* | Medium | Low |

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following four categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Fixed** The item has been confirmed and fixed by the client.

[2] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[3] https://cwe.mitre.org/

# Chapter 2 Findings

In total, we find **zero** potential issue. We have **one** recommendation and **two** notes.

- High Risk: 0
- Medium Risk: 0
- Low Risk: 0
- Recommendations: 1
- Notes: 2

| ID | Severity | Description | Category | Status |
|---|---|---|---|---|
| 1 | - | Inconsistency of Constant Name | Recommendation | Confirmed |
| 2 | - | Assumption on the Secure Implementation of NEAR SDK | Notes | Confirmed |
| 3 | - | Fixed Total Supply Owned by the Contract Owner | Notes | Confirmed |

The details are provided in the following sections.

## 2.1 Additional Recommendation

### 2.1.1 Inconsistency of Constant Name

**Status** Confirmed

**Introduced by** `Version 1`

**Description** The constant string name `SVG_PARAS_ICON` does not match the file type (PNG) of the `PARAS` icon.

```
34 const SVG_PARAS_ICON: &str = " ...
```

**Listing 2.1:** ft/src/lib.rs

**Suggestion** It is suggested to change the constant name from `SVG_PARAS_ICON` to `PNG_PARAS_ICON`.

## 2.2 Notes

### 2.2.1 Assumption on the Secure Implementation of NEAR SDK

**Status** Confirmed

**Introduced by** `Version 1`

**Description** This `PARAS` token contract is built based on the `near-sdk` (version 4.0.0-pre.7). The required interfaces and the basic functionality for NEP-141 (Fungible Token Standard), NEP-145 (Storage Management Standard) and NEP-148 (Fungible Token Metadata Standard) are provided in the token contract. In this audit, we assume the standard library provided by NEAR SDK (i.e., near_contract_standards) has no security issues.

```
18  use near_contract_standards::fungible_token::metadata::{
19      FungibleTokenMetadata, FungibleTokenMetadataProvider, FT_METADATA_SPEC,
20  };
21  use near_contract_standards::fungible_token::FungibleToken;
```

**Listing 2.2:** ft/src/lib.rs

```
90  near_contract_standards::impl_fungible_token_core!(Contract, token, on_tokens_burned);
91  near_contract_standards::impl_fungible_token_storage!(Contract, token, on_account_closed);
```

**Listing 2.3:** ft/src/lib.rs

**Feedback from the Project** Confirmed, it is based on NEAR SDK.

### 2.2.2 Fixed Total Supply Owned by the Contract Owner

**Status** Confirmed

**Introduced by** Version 1

**Description** The contract can be initialized only once and the given total supply will be owned by the given account ID. By default, the total supply of the token PARAS is 100_000_000_000_000_000_000_000_000 (with decimals 18), and the token cannot be minted or burnt after the initialization.

```
35 const TOTAL_SUPPLY: Balance = 100_000_000_000_000_000_000_000_000;
36
37 #[near_bindgen]
38 impl Contract {
39     #[init]
40     pub fn new_paras_meta(owner_id: AccountId) -> Self {
41         Self::new(
42             owner_id,
43             U128(TOTAL_SUPPLY),
44             FungibleTokenMetadata {
45                 spec: FT_METADATA_SPEC.to_string(),
46                 name: "PARAS".to_string(),
47                 symbol: "PARAS".to_string(),
48                 icon: Some(SVG_PARAS_ICON.to_string()),
49                 reference: None,
50                 reference_hash: None,
51                 decimals: 18
52             },
53         )
54     }
55
56     /// Initializes the contract with the given total supply owned by the given `owner_id` with
57     /// the given fungible token metadata.
58     #[init]
59     pub fn new(
60         owner_id: AccountId,
61         total_supply: U128,
62         metadata: FungibleTokenMetadata,
63     ) -> Self {
```

```
64        assert!(!env::state_exists(), "Already initialized");
65        metadata.assert_valid();
66        let mut this = Self {
67            token: FungibleToken::new(b"a".to_vec()),
68            metadata: LazyOption::new(b"m".to_vec(), Some(&metadata)),
69        };
70        this.token.internal_register_account(&owner_id);
71        this.token.internal_deposit(&owner_id, total_supply.into());
72        near_contract_standards::fungible_token::events::FtMint {
73            owner_id: &owner_id,
74            amount: &total_supply,
75            memo: Some("Initial tokens supply is minted"),
76        }
77            .emit();
78        this
79    }
```

**Listing 2.4:** ft/src/lib.rs

**Feedback from the Project** Confirmed, the fixed total supply is intended.