
# Security Audit Report for NEP141-Token-Convertor

**Date:** May 16st, 2022

**Version:** 1.0

**Contact**: contact@blocksec.com

BlockSec
Audit Team

Digitally signed by
BlockSec Audit Team
Date: 2022.05.16
13:49:37 +08'00'

# Contents

## Report Manifest

| Item | Description |
| --- | --- |
| Client | Octopus Network |
| Target | NEP141-Token-Convertor |

## Version History

| Version | Date | Description |
| --- | --- | --- |
| 1.0 | May 16th, 2022 | First Release |

**About BlockSec** The BlockSec Team focuses on the security of the blockchain ecosystem, and collaborates with leading DeFi projects to secure their products. The team 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 released detailed analysis reports of high-impact security incidents. 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 NEP141-Token-Convertor [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 | Version | Commit SHA |
|---|---|---|
| NEP141-Token-Convertor | `Version 1` | `7c8a44b49bf62ee57656f67a0ebcea55cb8015c5` |
| | `Version 2` | `c2bc82609de2c5f89bd005e3dcc094f782ba7d53` |

Note that, we did **NOT** audit all the modules in the repository. The modules covered by this audit report include contracts under the **nep141-token-convertor-contract** folder. Specifically, the files covered in this audit include:

nep141-token-convertor-contract/src:

- account.rs
- constants.rs
- contract_viewers.rs
- external_trait.rs
- storage_impl.rs
- types.rs
- admin.rs
- contract_interfaces.rs
- conversion_pool.rs
- lib.rs
- token_receiver.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.

[1] https://github.com/octopus-network/nep141-token-convertor

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.

- **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
- Access control
- 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.

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

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

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

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

# Chapter 2 Findings

In total, we find 8 potential issues in the smart contract. We also have 9 recommendations, as follows:

- High Risk: 2
- Medium Risk: 1
- Low Risk: 5
- Recommendations: 9

| ID | Severity | Description | Category | Status |
|---|---|---|---|---|
| 1 | Low | *Insecure Project Configuration* | Software Security | Fixed |
| 2 | Low | *Contract cannot be Paused Compeletely* | Software Security | Fixed |
| 3 | Low | *Improper Numeric Type* | Software Security | Fixed |
| 4 | High | *Incorrect Calculation of Output Token Amount (I)* | DeFi Security | Fixed |
| 5 | High | *Incorrect Calculation of Output Token Amount (II)* | DeFi Security | Fixed |
| 6 | Low | *Improper Storage Management* | DeFi Security | Fixed |
| 7 | Low | *Missing Configuration Check* | DeFi Security | Fixed |
| 8 | Medium | *Accounts cannot be Registered in ft_transfer_resolved* | DeFi Security | Fixed |
| 9 | - | *Potential Centralization Problem* | Recommendation | Confirmed |
| 10 | - | *Potential Elastic Supply Token Issue* | Recommendation | Confirmed |
| 11 | - | *Unnecessary Macro Decoration (I)* | Recommendation | Fixed |
| 12 | - | *Unnecessary Macro Decoration (II)* | Recommendation | Fixed |
| 13 | - | *Gas Optimization* | Recommendation | Fixed |
| 14 | - | *Redundant Code (I)* | Recommendation | Fixed |
| 15 | - | *Redundant Code (II)* | Recommendation | Fixed |
| 16 | - | *Potential DoS Problem* | Recommendation | Fixed |
| 17 | - | *Code and Runtime Optimization* | Recommendation | Fixed |

The details are provided in the following sections.

## 2.1 Software Security

### 2.1.1 Insecure Project Configuration

**Status** Fixed in `version 2`

**Introduced by** `version 1`

**Description** By default, the Rust programming language does not check integer overflow/underflow during the runtime, resulting in unexpected behaviors.

```
1   [package]
2   name = "nep141-token-convertor-contract"
3   version = "0.1.0"
4   edition = "2018"
5
6   [lib]
7   crate-type = ["cdylib", "rlib"]
8
9   [dependencies]
10  near-sdk = "4.0.0-pre.7"
```

```
11  near-contract-standards = "4.0.0-pre.7"
12  uint = { version = "0.9.0", default-features = false }
13  itertools = "0.10.0"
14
15  [dev-dependencies]
16  near-sdk-sim = "4.0.0-pre.7"
17  test-token = {path = "../test-token"}
```

**Listing 2.1:** nep141-token-convertor-contract/Cargo.toml

**Impact** Without enabling the runtime check, integer overflow or underflow may introduce unexpected behaviors.

**Suggestion I** Enable the runtime overflow check for the release target in this project's configuration.

### 2.1.2 Contract cannot be Paused Compeletely

**Status** Fixed in `version 2`

**Introduced by** `version 1`

**Description** Assertion `assert_contract_is_not_paused` is missing in some key functions.

```
81  pub(crate) fn assert_contract_is_not_paused(&self) {
82      assert!(!self.contract_is_paused, "contract is paused")
83  }
```

**Listing 2.2:** nep141-token-convertor-contract/src/lib.rs

These functions are listed below:

- storage_deposit
- storage_withdraw
- storage_unregister

**Impact** Users can still register or unregister accounts when the contract is paused by the administrator.

**Suggestion I** Add assertion `assert_contract_is_not_paused` in these functions.

### 2.1.3 Improper Numeric Type

**Status** Fixed in `version 2`

**Introduced by** `version 1`

**Description** The `pool_id` is increased by 1 every time a new pool is created, which is shown in function `internal_assign_pool_id`. The type `u32` may not be enough for `pool_id` as the maximum number of `u32` is $2^{32} - 1 = 4,294,967,295$.

```
229  pub(crate) fn internal_assign_pool_id(&mut self) -> u32 {
230      self.pool_id += 1;
231      return self.pool_id;
232  }
```

**Listing 2.3:** nep141-token-convertor-contract/src/conversion_pool.rs

**Impact** The pool_id is likely to overflow with huge number of pools created.

**Suggestion I** Change the numeric type of `pool_id` to be a larger one and add the overflow check.

## 2.2 DeFi Security

### 2.2.1 Incorrect Calculation of Output Token Amount (I)

**Status** Fixed in `version 2`

**Introduced by** `version 1`

**Description** The `output_token_amount` will be equal to the `input_token_amount` if we multiply and then divide the same token rate `self.in_token_rate`.

```
115    pub fn calculate_output_token_amount(&self, token_amount: Balance) -> Balance {
116        (U256::from(token_amount) * U256::from(self.in_token_rate) / U256::from(self.in_token_rate)
               )
117            .as_u128()
118    }
```

**Listing 2.4:** nep141-token-convertor-contract/src/conversion_pool.rs

**Impact** Users may receive an unexpected amount of DeFi assets.

**Suggestion I** Fix the formula for the convertion in function `calculate_output_token_amount`.

### 2.2.2 Incorrect Calculation of Output Token Amount (II)

**Status** Fixed in `version 2`

**Introduced by** `version 1`

**Description** In this function, the multiplier `self.out_token_rate` and divisor `self.in_token_rate` are reversed, which is contrary to the price mechanism of the Nep141-Token-Converter.

```
120    pub fn calculate_reverse_output_token_amount(&self, token_amount: Balance) -> Balance {
121        (U256::from(token_amount) * U256::from(self.out_token_rate)
122            / U256::from(self.in_token_rate))
123        .as_u128()
124    }
```

**Listing 2.5:** nep141-token-convertor-contract/src/conversion_pool.rs

**Impact** Users may receive an unexpected amount of DeFi assets.

**Suggestion I** Fix the formula for the convertion in function `calculate_reverse_output_token_amount`.

### 2.2.3 Improper Storage Management

**Status** Fixed in `version 2`

**Introduced by** `version 1`

**Description** The `account_id` in function `assert_storage_balance_bound_min` may not be equal to the `env::predecessor_account_id()`, which represents the previous account in the chain of cross-contract calls. For example, the `env::predecessor_account_id()` can also be the admin account. Therefore, it is unfair to compare the near amount for storage between the account specified by `account_id` and the `env::predecessor_account_id()`(lines 90-91).

```
85    pub(crate) fn assert_storage_balance_bound_min(&self, account_id: &AccountId) {
86        let account = self
87            .internal_get_account(account_id)
88            .expect(format!("user {} hasn't registered.", &env::predecessor_account_id()).as_str())
                  ;
89        assert!(
90            account.near_amount_for_storage
91                >= self.internal_get_storage_balance_min_bound(&env::predecessor_account_id()),
92            "Need deposit {} for storage.",
93            self.internal_get_storage_balance_min_bound(&env::predecessor_account_id())
94                - account.near_amount_for_storage
95        );
96    }
```

**Listing 2.6:** nep141-token-convertor-contract/src/lib.rs

**Impact** Users may fail to receive tokens due to storage fee comparison errors.

**Suggestion I** Change the parameter of function `internal_get_storage_balance_min_bound` into `account_id` instead of the `&env::predecessor_account_id()`.

### 2.2.4 Missing Configuration Check

**Status** Fixed in `version 2`

**Introduced by** `version 1`

**Description** Both the `in_token_rate` and `out_token_rate` should not be zero when a new `ConversionPool` is created.

```
47    pub fn new(
48        id: u32,
49        creator: AccountId,
50        in_token: AccountId,
51        out_token: AccountId,
52        reversible: bool,
53        in_token_rate: u32,
54        out_token_rate: u32,
55        deposit_near_amount: U128,
56    ) -> Self {
57        Self {
58            id,
59            creator,
60            in_token,
61            in_token_balance: U128(0),
62            out_token,
63            out_token_balance: U128(0),
64            reversible,
65            in_token_rate,
66            out_token_rate,
67            deposit_near_amount,
68        }
69    }
```

**Listing 2.7:** nep141-token-convertor-contract/src/conversion_pool.rs

**Impact** The pool may be out of service due to the division by zero error or the calculated `output_token_amount` may always be zero.

**Suggestion I** Add a proper assertion in function `new()` of struct `ConversionPool` for `in_token_rate` and `out_token_rate`.

### 2.2.5 Accounts cannot be Registered in ft_transfer_resolved

**Status** Fixed in `version 2`

**Introduced by** `version 1`

**Description** The function `ft_transfer_resolved` is used to rollback the state if the token is not transferred successfully. However, the account (receiver) cannot be registered in `ft_transfer_resolved` due to the storage check in `internal_save_account` if the `PromiseResult` is failed.

```
106    #[private]
107    pub fn ft_transfer_resolved(
108        &mut self,
109        token_id: AccountId,
110        sender_id: AccountId,
111        amount: U128,
112    ) {
113        assert_eq!(
114            env::promise_results_count(),
115            1,
116            "expected 1 promise result from withdraw"
117        );
118        match env::promise_result(0) {
119            PromiseResult::NotReady => unreachable!(),
120            PromiseResult::Successful(_) => {}
121            PromiseResult::Failed => {
122                // This reverts the changes from withdraw function.
123                // If account doesn't exit, deposits to the owner's account as lostfound.
124
125                log!("Transfer token failed.Try to deposit token into account.");
126                let mut account = self
127                    .internal_get_account(&sender_id)
128                    .unwrap_or(Account::new());
129                account.deposit_token(&token_id, amount.0);
130                self.internal_save_account(&sender_id, account);
131            }
132        };
133    }
```

**Listing 2.8:** nep141-token-convertor-contract/src/token_receiver.rs

**Impact** The token is locked in this contract.

**Suggestion I** Ensure that the account (receiver) exists when the `ft_transfer_resolved` is executed.

## 2.3 Additional Recommendation

### 2.3.1 Potential Centralization Problem

**Status** Confirmed

**Introduced by** `version 1`

**Description** This project has potential centralization problems and the `TokenConvertor.admin` cannot be changed since the contract is deployed and initialized.

The project owner needs to ensure the security of the private key of the `TokenConvertor.admin` and use a multi-signature scheme to reduce the risk of single-point failure.

**Suggestion I** It is recommended to introduce a decentralization design in the contract, such as multi-signature or DAO.

### 2.3.2 Potential Elastic Supply Token Issue

**Status** Confirmed

**Introduced by** `version 1`

**Description** Elastic supply tokens (e.g., deflation tokens) could dynamically adjust the supply or user's balance. For example, if the token is a deflation token, there will be a difference between the transferred amount of tokens and the actual received amount of tokens.

This inconsistency can lead to security impacts for the operations based on the transferred amount of tokens instead of the actual received amount of tokens.

**Suggestion I** Do not append the elastic supply tokens into the whitelist.

### 2.3.3 Unnecessary Macro Decoration (I)

**Status** Fixed in `version 2`

**Introduced by** `version 1`

**Description** It's unnecessary to decorate functions `internal_get_pool` and `internal_save_pool` with macro `private` since they are already internal functions.

```
266     #[private]
267     pub(crate) fn internal_get_pool(&self, pool_id: &PoolId) -> Option<ConversionPool> {
268         // return self.pools.get(&pool_id)
269         return self.pools.get(pool_id).map(|pool| pool.into_current());
270     }
271
272     #[private]
273     pub(crate) fn internal_save_pool(&mut self, pool_id: PoolId, pool: &VPool) {
274         self.pools.insert(&pool_id, &pool);
275         // self.pools.replace(pool_id, &pool);
276     }
```

**Listing 2.9:** nep141-token-convertor-contract/src/conversion_pool.rs

**Suggestion I** Remove the macro `private` for functions `internal_get_pool` and `internal_save_pool`.

### 2.3.4 Unnecessary Macro Decoration (II)

**Status** Fixed in `version 2`

**Introduced by** `version 1`

**Description** It's unnecessary to add macro `near_bindgen` for the implementation, which is listed below, if all the functions defined in are internal functions.

```
216   #[near_bindgen]
217   impl TokenConvertor {
218     pub(crate) fn internal_convert(
```

**Listing 2.10:** nep141-token-convertor-contract/src/conversion_pool.rs

```
78   #[near_bindgen]
79   impl TokenConvertor {
80     pub(crate) fn internal_get_account(&self, account_id: &AccountId) -> Option<Account> {
```

**Listing 2.11:** nep141-token-convertor-contract/src/account.rs

**Suggestion I** Remove the macro `near_bindgen` for the implementations listed above.

### 2.3.5 Gas Optimization

**Status** Fixed in `version 2`

**Introduced by** `version 1`

**Description** Gas consumption can be optimized in three loactions:

Loaction I:

`internal_get_storage_balance_min_bound` is invoked twice in function `storage_deposit`. Line 29 can be replaced with the variable `min_balance`.

```
12   #[payable]
13   fn storage_deposit(
14       &mut self,
15       account_id: Option<AccountId>,
16       registration_only: Option<bool>,
17   ) -> StorageBalance {
18       let attach_amount = env::attached_deposit();
19       let account_id = account_id.unwrap_or(env::predecessor_account_id());
20       let mut account = self
21           .internal_get_account(&account_id)
22           .unwrap_or(Account::new());
23       let registration_only = registration_only.unwrap_or(false);
24       let min_balance = self.internal_get_storage_balance_min_bound(&account_id);
25       log!(
26           "{} storage deposit {} yocto near, storage_balance_bounds.min is {}",
27           env::predecessor_account_id(),
28           env::attached_deposit(),
29           self.internal_get_storage_balance_min_bound(&account_id)
30
31       );
```

**Listing 2.12:** contracts/linear/src/storage_impl.rs

Loaction II:

`internal_get_storage_balance_min_bound` is invoked twice in function `get_storage_fee_gap_of`. This can be optimized by introducing one local variable.

```
85pub(crate) fn assert_storage_balance_bound_min(&self, account_id: &AccountId) {
86    let account = self
87        .internal_get_account(account_id)
88        .expect(format!("user {} hasn't registered.", &env::predecessor_account_id()).as_str());
89    assert!(
90        account.near_amount_for_storage
91            >= self.internal_get_storage_balance_min_bound(&env::predecessor_account_id()),
92        "Need deposit {} for storage.",
93        self.internal_get_storage_balance_min_bound(&env::predecessor_account_id())
94            - account.near_amount_for_storage
95    );
96}
```

**Listing 2.13:** nep141-token-convertor-contract/src/lib.rs

Loaction III:

This location has the same problem as mentioned above.

```
12    fn get_storage_fee_gap_of(&self, account_id: AccountId) -> U128 {
13        let near_amount_for_storage = self
14            .internal_get_account(&account_id)
15            .map(|e| e.near_amount_for_storage)
16            .unwrap_or(0);
17        return if near_amount_for_storage
18            >= self.internal_get_storage_balance_min_bound(&account_id)
19        {
20            U128(0)
21        } else {
22            U128(self.internal_get_storage_balance_min_bound(&account_id) - near_amount_for_storage
                  )
23        };
24    }
```

**Listing 2.14:** contracts/linear/src/storage_impl.rs

**Suggestion I** Optimize the gas consumption.

### 2.3.6 Redundant Code (I)

**Status** Fixed in `version 2`

**Introduced by** `version 1`

**Description** Dead Code found in module `types`.

```
12    #[derive(BorshSerialize, BorshDeserialize)]
13    pub struct TokenDirectionKey(String);
14
15    impl TokenDirectionKey {
16        pub fn new(from_token: &AccountId, to_token: &AccountId) -> Self {
17            Self {
```

```
18                 0: format!("{}:{}", from_token.to_string(), to_token.to_string()),
19             }
20         }
21     }
```

**Listing 2.15:** nep141-token-convertor-contract/src/types.rs

**Suggestion I** Remove the dead code.

### 2.3.7 Redundant Code (II)

**Status** Fixed in `version 2`

**Introduced by** `version 1`

**Description** Re-calculation of gas fee in function `internal_send_tokens` in line 94 and 102 since they are already saved in local variables `ft_transfer_gas` and `ft_transfer_resolved_gas` defined in lines 85-86.

```
78     pub(crate) fn internal_send_tokens(
79         &self,
80         receiver_id: &AccountId,
81         token_id: &AccountId,
82         amount: Balance,
83     ) -> Promise {
84         self.assert_storage_balance_bound_min(receiver_id);
85         let ft_transfer_gas = Gas::ONE_TERA.mul(T_GAS_FOR_FT_TRANSFER);
86         let ft_transfer_resolved_gas = Gas::ONE_TERA.mul(T_GAS_FOR_RESOLVE_TRANSFER);
87         self.assert_remind_gas_greater_then(ft_transfer_gas + ft_transfer_resolved_gas);
88         ext_fungible_token::ft_transfer(
89             receiver_id.clone(),
90             U128(amount),
91             None,
92             token_id.clone(),
93             1,
94             Gas::ONE_TERA.mul(T_GAS_FOR_FT_TRANSFER),
95         )
96         .then(ext_self::ft_transfer_resolved(
97             token_id.clone(),
98             receiver_id.clone(),
99             U128(amount),
100            env::current_account_id(),
101            0,
102            Gas::ONE_TERA.mul(T_GAS_FOR_RESOLVE_TRANSFER),
103        ))
104    }
```

**Listing 2.16:** nep141-token-convertor-contract/src/token_receiver.rs

**Suggestion I** This can be optimized by using the local variables instead of re-calculation of the gas fee.

### 2.3.8 Potential DoS Problem

**Status** Fixed in `version 2`

**Introduced by** `version 1`

**Description** Users may create lots of `ConversionPools` without paying any extra storage fee if the `TokenConvertor.create_pool_deposit` is kept as the default value 0.

```
51    #[init]
52    pub fn new(admin: AccountId) -> Self {
53        Self {
54            admin,
55            accounts: LookupMap::new(StorageKey::Accounts),
56            pools: UnorderedMap::new(StorageKey::Pools),
57            whitelisted_tokens: UnorderedMap::new(StorageKey::WhitelistedTokens),
58            create_pool_deposit: 0,
59            pool_id: 0,
60            contract_is_paused: false,
61        }
62    }
```

**Listing 2.17:** nep141-token-convertor-contract/src/lib.rs

**Suggestion I** The owner needs to reset the `TokenConvertor.create_pool_deposit` timely with a reasonable value after the contract is initialized.

### 2.3.9 Code and Runtime Optimization

**Status** Fixed in `version 2`

**Introduced by** `version 1`

**Description** It is recommended to add these configurations listed below for the release target to optimize the code generation and runtime efficiency.

```
1    [profile.release]
2    codegen-units = 1
3    # s = optimize for binary size ("z" would additionally turn off loop vectorization)
4    opt-level = "s"
5    # link time optimization
6    lto = true
7    debug = false
8    panic = "abort"
9    overflow-checks = true
```

**Suggestion I** Optimize the code generation and runtime efficiency with suggested configuration.