BLOCKSEC

Security Audit
Report for TKN

Date: August 12, 2024 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	TKN Homes
Target	TKN

Version History

Version	Date	Description
1.0	August 12, 2024	First release

Signature

Digitally signed by: BlockSec (https://blocksec.com)

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2024-08-14 07:04:25+0000

About BlockSec 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 14 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 target of this audit is the code repository of tknhomesdev [1] of TKN Homes. Note that, we did **NOT** audit all the modules in the repository. The modules covered by this audit report include `tknhomesdev/contracts` folder contract only. Specifically, the files covered in this audit include:

```
1 common/src/deflation.rs
2 common/src/lib.rs
3 common/src/tknx_metadata.rs
4 common/src/utils.rs
5 factory/src/lib.rs
```

Listing 1.1: Audit Scope for this Report

The auditing process is iterative. Specifically, we would 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 table. Our audit report is responsible for the code in the initial version (`Version 1`), as well as new code (in the following versions) to fix issues in the audit report.

Project	Version	Commit Hash
TKN	`Version 1`	`55acff27de3aef7048605aed38aa98a68f348586`
	`Version 2`	`a14a8c53af8cb261d15a50a6be5f6d33d97528da`

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

[1] https://github.com/tknhomesdev/Contracts

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
* 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:

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

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

- **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.

Chapter 2 Findings

In total, we found **two** potential security issues. Besides, we have **three** recommendations and **three** notes.

- High Risk: 0
- Medium Risk: 1
- Low Risk: 1
- Recommendation: 3
- Note: 3

ID	Severity	Description	Category	Status
1	Medium	Lack of strategy check in function `create_token()`	DeFi Security	Fixed
2	Low	Inconsistency between documentation and implementation	DeFi Security	Fixed
3	-	Lack of check on the removed `white_account`	Recommendation	Fixed
4	-	Redundant check on `args.metadata`	Recommendation	Confirmed
5	-	Potential duplicated `white_account`	Recommendation	Fixed
6	-	Potential zero deflation fee due to precision loss	Note	-
7	-	Tokens not actually burned during transfers	Note	-
8	-	Potential centralization risk	Note	-

The details are provided in the following sections.

2.1 DeFi Security

2.1.1 Lack of strategy check in function `create_token()`

Severity Medium

Status Fixed at `Version 2`

Introduced by `Version 1`

Description In the function `create_token()` of the `tokenfactory` contract, it does not check whether the `args.deflation_strategy` is valid. Meanwhile, the function `new()` of the token contract checks if the `deflation_strategy` is valid. Therefore, if a user provides an incorrect `deflation_strategy`, the function `new()` in the token contract will fail, and there is no corresponding callback function in `create_token()` to handle this situation, which is incorrect.

```
144pub fn create_token(&mut self, args: TokenArgs) -> Promise {
145   if !env::attached_deposit().is_zero() {
146       self.storage_deposit();
147   }
148   args.metadata.assert_valid();
149   assert!(args.protocol_account_id.is_none());
```

```
150   assert!(args.burn_account_id.is_none());
151   let args = self.update_args(args);
152   let token_id = args.metadata.symbol.to_ascii_lowercase();
153   assert!(is_valid_token_id(&token_id), "Invalid Symbol");
154   let token_account_id = format!("{}.{}", token_id, env::current_account_id());
155   assert!(
156       env::is_valid_account_id(token_account_id.as_bytes()),
157       "Token Account ID is invalid"
158   );
159
160
161   let account_id = env::predecessor_account_id();
162
163
164   let required_balance = self.get_min_attached_balance(&args);
165   let user_balance = self.storage_deposits.get(&account_id).unwrap_or(0);
166   assert!(
167       user_balance >= required_balance,
168       "Not enough required balance"
169   );
170   self.storage_deposits
171       .insert(&account_id, &(user_balance - required_balance));
172
173
174   let initial_storage_usage = env::storage_usage();
175
176
177   assert!(
178       self.tokens.insert(&token_id, &args).is_none(),
179       "Token ID is already taken"
180   );
181
182
183   let storage_balance_used =
184       env::storage_byte_cost().checked_mul((env::storage_usage() - initial_storage_usage) as u128)
            .unwrap().as_yoctonear();
185
186
187   Promise::new(token_account_id.parse().unwrap())
188       .create_account()
189       .transfer(NearToken::from_yoctonear(required_balance - storage_balance_used))
190       .deploy_contract(FT_WASM_CODE.to_vec())
191       .function_call("new".to_string(), serde_json::to_vec(&args).unwrap(), NearToken::
            from_yoctonear(0), GAS)
192}
```

Listing 2.1: contracts/factory/src/lib.rs

```
64pub fn new(
65   owner_account_id: AccountId,
66   total_supply: U128,
67   metadata: FungibleTokenMetadata,
68
```

```
69
70  protocol_account_id: AccountId,
71  burn_account_id: AccountId,
72  deflation_strategy: DeflationStrategy,
73
74
75  fee_strategy_white_list: Vec<AccountId>,
76  burn_strategy_white_list: Vec<AccountId>,
77) -> Self {
78  require!(!env::state_exists(), "Already initialized");
79  metadata.assert_valid();
80  deflation_strategy.assert_valid();
81  let mut this = Self {
82      token: FungibleToken::new(StorageKey::FungibleToken),
83      metadata: LazyOption::new(StorageKey::Metadata, Some(&metadata)),
84
85
86      owner_account_id: owner_account_id.clone(),
87      protocol_account_id: protocol_account_id.clone(),
88      burn_account_id: burn_account_id.clone(),
89
90
91      deflation_strategy,
92      accumulated_info: Default::default(),
93      fee_strategy_white_list: UnorderedSet::new(StorageKey::FeeStrategyWhiteList),
94      burn_strategy_white_list: UnorderedSet::new(StorageKey::BurnStrategyWhiteList),
95  };
96  this.token.internal_register_account(&owner_account_id);
97  this.token.internal_register_account(&protocol_account_id);
98  this.token.internal_register_account(&burn_account_id);
99  this.token.internal_deposit(&owner_account_id, total_supply.into());
100
101
102 for account_id in fee_strategy_white_list.iter() {
103     this.fee_strategy_white_list.insert(account_id);
104 }
105
106
107 for account_id in burn_strategy_white_list.iter() {
108     this.burn_strategy_white_list.insert(account_id);
109 }
110
111
112 near_contract_standards::fungible_token::events::FtMint {
113     owner_id: &owner_account_id,
114     amount: total_supply,
115     memo: Some("new tokens are minted"),
116 }
117 .emit();
118
119
120 this
121}
```

Listing 2.2: contracts/common/src/lib.rs

Impact If the execution of the function `new()` in the token contract fails, the `NEAR` sent during initialization cannot be refunded, and users cannot create the token with the same `symbol`.

Suggestion Add a validation step in the function `create_token()` to ensure that the `deflation_strategy` provided by the user is valid.

2.1.2 Inconsistency between documentation and implementation

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the file `deflation.rs`, according to the comment on line 34 (i.e., "Will charge a fee if the target accountId is in the white list,"), a `TransferFee` will be charged for a `receiver_id` that is in the white list. However, according to the implementation of the function `skip_strategy()`, if the `receiver_id` is in the white list, it returns true. Consequently, the check `fee_strategy.skip_strategy(self.fee_strategy_white_list.contains(receiver_id))` on line 130 will return false, and no fee will be charged. The same issue also exists in `BurnStrategy`.

```
33  pub enum FeeStrategy {
34  /// Will charge fee if the target accountId is in white list.
35  TransferFee { fee_rate: u32 },
36  /// Will charge fee if the target accountId is NOT in white list.
37  SellFee { fee_rate: u32 }
38}
```

Listing 2.3: contracts/common/src/deflation.rs

```
53  pub fn skip_strategy(&self, is_white_list_account: bool) -> bool {
54      match self {
55          FeeStrategy::TransferFee { .. } => is_white_list_account,
56          FeeStrategy::SellFee { .. } => !is_white_list_account,
57      }
58  }
```

Listing 2.4: contracts/common/src/deflation.rs

```
124 pub fn internal_calculate_deflation_detail(&self, sender_id: &AccountId, receiver_id: &
        AccountId, amount: u128) -> DeflationDetail {
125     let mut deflation_detail: DeflationDetail = Default::default();
126     if !self.deflation_strategy.is_deflation_mode() || sender_id == &self.owner_account_id {
127         return deflation_detail;
128     }
129     if let Some(ref fee_strategy) = self.deflation_strategy.fee_strategy {
130         if !fee_strategy.skip_strategy(self.fee_strategy_white_list.contains(receiver_id)) {
131             deflation_detail.fee_amount = fee_strategy.deflation(amount);
132         }
133     }
```

```
134         if let Some(ref burn_strategy) = self.deflation_strategy.burn_strategy {
135             if !burn_strategy.skip_strategy(self.burn_strategy_white_list.contains(receiver_id)) {
136                 deflation_detail.burn_amount = burn_strategy.deflation(amount);
137             }
138         }
139         deflation_detail
140     }
```

Listing 2.5: contracts/common/src/deflation.rs

Impact The code implementation does not match the comment.

Suggestion Revise the logic to match the implementation and comment.

2.2 Additional Recommendation

2.2.1 Lack of check on the removed `white_account`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description When using the function `remove_protocol_fee_strategy_white_list()` to remove a `white_account` from the `protocol_burn_strategy_white_list`, there is no check on whether the account to be removed exists in the `protocol_burn_strategy_white_list`. The same issue also exists in the function `remove_protocol_burn_strategy_white_list()`.

```
207     #[payable]
208     pub fn remove_protocol_fee_strategy_white_list(&mut self, white_account: AccountId) {
209         self.assert_owner_with_yocto();
210         if let Some(protocol_fee_strategy_white_list) = self.protocol_fee_strategy_white_list.
                as_mut(){
211             protocol_fee_strategy_white_list.retain(|a| a != &white_account);
212         }
213     }
```

Listing 2.6: contracts/factory/src/lib.rs

```
214     #[payable]
215     pub fn remove_protocol_burn_strategy_white_list(&mut self, white_account: AccountId) {
216         self.assert_owner_with_yocto();
217         if let Some(protocol_burn_strategy_white_list) = self.protocol_burn_strategy_white_list.
                as_mut(){
218             protocol_burn_strategy_white_list.retain(|a| a != &white_account);
219         }
220     }
```

Listing 2.7: contracts/factory/src/lib.rs

Suggestion Add relevant checks accordingly.

2.2.2 Redundant check on `args.metadata`

Status Confirmed

Introduced by `Version 1`

Description The check on line 77 in the function `new()` of the token contract is unnecessary. Specifically, the function `create_token()` in the `tokenfactory` contract has already checked the validation of `metadata`. Therefore, the check of `metadata` in the function `new()` of the token contract is redundant.

```
144 pub fn create_token(&mut self, args: TokenArgs) -> Promise {
145   if !env::attached_deposit().is_zero() {
146       self.storage_deposit();
147   }
148   args.metadata.assert_valid();
149   assert!(args.protocol_account_id.is_none());
150   assert!(args.burn_account_id.is_none());
151   let args = self.update_args(args);
152   let token_id = args.metadata.symbol.to_ascii_lowercase();
153   assert!(is_valid_token_id(&token_id), "Invalid Symbol");
154   let token_account_id = format!("{}.{}", token_id, env::current_account_id());
155   assert!(
156       env::is_valid_account_id(token_account_id.as_bytes()),
157       "Token Account ID is invalid"
158   );
159
160
161   let account_id = env::predecessor_account_id();
162
163
164   let required_balance = self.get_min_attached_balance(&args);
165   let user_balance = self.storage_deposits.get(&account_id).unwrap_or(0);
166   assert!(
167       user_balance >= required_balance,
168       "Not enough required balance"
169   );
170   self.storage_deposits
171       .insert(&account_id, &(user_balance - required_balance));
172
173
174   let initial_storage_usage = env::storage_usage();
175
176
177   assert!(
178       self.tokens.insert(&token_id, &args).is_none(),
179       "Token ID is already taken"
180   );
181
182
183   let storage_balance_used =
184       env::storage_byte_cost().checked_mul((env::storage_usage() - initial_storage_usage) as u128)
            .unwrap().as_yoctonear();
185
```

```
186
187   Promise::new(token_account_id.parse().unwrap())
188       .create_account()
189       .transfer(NearToken::from_yoctonear(required_balance - storage_balance_used))
190       .deploy_contract(FT_WASM_CODE.to_vec())
191       .function_call("new".to_string(), serde_json::to_vec(&args).unwrap(), NearToken::
              from_yoctonear(0), GAS)
192}
```

Listing 2.8: contracts/factory/src/lib.rs

```
64 pub fn new(
65   owner_account_id: AccountId,
66   total_supply: U128,
67   metadata: FungibleTokenMetadata,
68
69
70   protocol_account_id: AccountId,
71   burn_account_id: AccountId,
72   deflation_strategy: DeflationStrategy,
73
74
75   fee_strategy_white_list: Vec<AccountId>,
76   burn_strategy_white_list: Vec<AccountId>,
77) -> Self {
78   require!(!env::state_exists(), "Already initialized");
79   metadata.assert_valid();
80   deflation_strategy.assert_valid();
81   let mut this = Self {
82       token: FungibleToken::new(StorageKey::FungibleToken),
83       metadata: LazyOption::new(StorageKey::Metadata, Some(&metadata)),
84
85
86       owner_account_id: owner_account_id.clone(),
87       protocol_account_id: protocol_account_id.clone(),
88       burn_account_id: burn_account_id.clone(),
89
90
91       deflation_strategy,
92       accumulated_info: Default::default(),
93       fee_strategy_white_list: UnorderedSet::new(StorageKey::FeeStrategyWhiteList),
94       burn_strategy_white_list: UnorderedSet::new(StorageKey::BurnStrategyWhiteList),
95   };
96   this.token.internal_register_account(&owner_account_id);
97   this.token.internal_register_account(&protocol_account_id);
98   this.token.internal_register_account(&burn_account_id);
99   this.token.internal_deposit(&owner_account_id, total_supply.into());
100
101
102   for account_id in fee_strategy_white_list.iter() {
103       this.fee_strategy_white_list.insert(account_id);
104   }
105
```

```
106
107  for account_id in burn_strategy_white_list.iter() {
108      this.burn_strategy_white_list.insert(account_id);
109  }
110
111
112  near_contract_standards::fungible_token::events::FtMint {
113      owner_id: &owner_account_id,
114      amount: total_supply,
115      memo: Some("new tokens are minted"),
116  }
117  .emit();
118
119
120  this
121}
```

Listing 2.9: contracts/factory/src/lib.rs

Suggestion Remove this redundant check.

Feedback from the project Token contracts might be deployed directly, so we choose to retain checks.

2.2.3 Potential duplicated `white_account`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description When using the function `add_protocol_fee_strategy_white_list()` to add a white list account to the `protocol_fee_strategy_white_list`, there is no check to see if the account already exists, which may result in duplicate white list accounts in the `protocol_fee_strategy_white_list`. It also exists in the function `add_protocol_burn_strategy_white_list()`.

```
188   #[payable]
189   pub fn add_protocol_fee_strategy_white_list(&mut self, white_account: AccountId) {
190       self.assert_owner_with_yocto();
191       if let Some(protocol_fee_strategy_white_list) = self.protocol_fee_strategy_white_list.
              as_mut(){
192           protocol_fee_strategy_white_list.push(white_account);
193       } else {
194           self.protocol_fee_strategy_white_list = Some(vec![white_account]);
195       }
196   }
```

Listing 2.10: contracts/factory/lib.rs

```
197   #[payable]
198   pub fn add_protocol_burn_strategy_white_list(&mut self, white_account: AccountId) {
199       self.assert_owner_with_yocto();
200       if let Some(protocol_burn_strategy_white_list) = self.protocol_burn_strategy_white_list.
              as_mut(){
```

```
201             protocol_burn_strategy_white_list.push(white_account);
202         } else {
203             self.protocol_burn_strategy_white_list = Some(vec![white_account]);
204         }
205     }
```

Listing 2.11: contracts/factory/lib.rs

Suggestion Add duplicate checks in function `add_protocol_fee_strategy_white_list()` and `add_protocol_burn_strategy_white_list()`.

2.3 Notes

2.3.1 Potential zero deflation fee due to precision loss

Introduced by `Version 1`

Description When calculating the deflation fee, it is possible that due to a very small transfer amount combined with rounding down used in the calculation, the deducted deflation fee ends up being 0.

2.3.2 Tokens not actually burned during transfers

Introduced by `Version 1`

Description According to the design, during the transfer process, the token contract will collect a certain percentage of deflation fee from the sender. Part of it is named as the `burn_amount`, but this is not actually a burn operation, but rather a transfer to the `burn_amount`. The address of this `burn_amount` is set by the deployer of the token.

2.3.3 Potential centralization risk

Introduced by `Version 1`

Description In the project, there is a privileged account `owner`, which can add and remove whitelist accounts, and upgrade contracts. If the `owner`'s private key is lost or maliciously exploited, it could potentially cause losses to the protocol and users.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS