Security Audit Report for Near-IBC

Date: Sep 18, 2023

Version: 1.0

Contact: contact@blocksec.com

BlockSec Audit Team

Digitally signed by BlockSec Audit Team
Date: 2023.09.18 17:14:49 +08'00'

Contents

Report Manifest

Item	Description
Client	Octopus Network
Target	Near-IBC

Version History

Version	Date	Description
1.0	September 18, 2023	First Version

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 near-ibc [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 (i.e., `Version 1`), as well as new codes (in the following versions) to fix issues in the audit report.

Project		Commit SHA
Near-IBC	`Version 1`	`aa0feefd4cdd6f7c707ef16cb3b3fb8cd4c1cb31`
	`Version 2`	`52afaaafa5891082419db551e5cf1dee0019a45a`

Note that, we did **NOT** audit all the modules in the repository. The modules covered by this audit report include **near-ibc-main** folder contract only. Specifically, the files covered in this audit include:

- channel-escrow
- escrow-factory
- near-ibc
- token-factory
- utils
- wrapped-token

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.

[1] https://github.com/octopus-network/near-ibc

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.

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 **twelve** potential issues. Besides, we have **two** recommendations and **four** notes as follows:

- High Risk: 4
- Medium Risk: 7
- Low Risk: 1
- Recommendations: 2
- Notes: 4

ID	Severity	Description	Category	Status
1	High	Incorrect Check of Ancestor Account	Software Security	Fixed
2	High	User-controllable Token Denom	Software Security	Fixed
3	High	Lack of Access Control in register_assets()	Software Security	Fixed
4	Medium	Lack of Handling for Failure Transfers	Software Security	Confirmed
5	High	Inconsistent Use of Storage Key	Software Security	Fixed
6	Medium	Improper Check of Attached NEAR Value	Defi Security	Fixed
7	Medium	Attached NEAR Repeatedly Used in Multiple Contracts	DeFi Security	Fixed
8	Medium	Lack of #[payable] Modifier for function register_asset_for_channel()	DeFi Security	Fixed
9	Medium	Incorrect Use of refund_deposit()	DeFi Security	Fixed
10	Medium	Lack of Implementation of Storage Check and Refund for register_asset()	DeFi Security	Fixed
11	Medium	Incorrect Results Returned during Executions	DeFi Security	Confirmed
12	Low	Improper Check in send_coins_execute()	DeFi Security	Fixed
13	-	Redundant Uninitialized Check	Recommendation	Fixed
14	-	Lack of Check on denom	Recommendation	Confirmed
15	-	Assumption on the Secure Implementation of Dependencies	Note	Confirmed
16	-	Storage Consumption when Relay Messages	Note	Confirmed
17	-	Source Chain Unavailable for Multiple Cross Chain Redemption	Note	Confirmed
18	-	Failed Retrieval of Host Consensus State	Note	Confirmed

The details are provided in the following sections.

2.1 Software Security

2.1.1 Incorrect Check of Ancestor Account

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The function `assert_ancestor_account()` is used by privileged functions to implement access control. Specifically, it requires the caller to be an "`ancestor`" account of the current contract ac-

count. In Near, an account can create sub-accounts, and this account is referred to as the ancestor account of the created sub-accounts. The name of these sub-accounts are named by extending the name of the ancestor account. For example, the sub-accountof "ancestor.near" can be named as "xxx.ancestor.near" or "yyy.xxx.ancestor.near" (where "xxx" and "yyy" can be any legal characters).

However, the current implementation is wrong as it tries to use the function ends_with() for the check. Taking the contract account "xxx.ancestor.near" as an example, after performing the split_once() operation, the parent becomes "ancestor.near". In this case, if the caller account is "ncestor.near", this check can also be bypassed.

The similar issue also exists in the function assert_sub_account().

```
110/// Asserts that the predecessor account is the sub account of current account.
111pub fn assert_sub_account() {
112    let account_id = env::predecessor_account_id().to_string();
113    let (_first, parent) = account_id.split_once(".").unwrap();
114    assert!(
115        parent.ends_with(env::current_account_id().as_str()),
116        "ERR_ONLY_SUB_ACCOUNT_CAN_CALL_THIS_METHOD"
117    );
118}
119
120/// Asserts that the predecessor account is an ancestor account of current account.
121pub fn assert_ancestor_account() {
122    let account_id = env::current_account_id().to_string();
123    let (_first, parent) = account_id.split_once(".").unwrap();
124    assert!(
125        parent.ends_with(env::predecessor_account_id().as_str()),
126        "ERR_ONLY_UPPER_LEVEL_ACCOUNT_CAN_CALL_THIS_METHOD"
127    );
128}
```

Listing 2.1: utils/src/lib.rs

Impact All privileged functions that implement access control using the above functions can be bypassed.

Suggestion Revise the logic accordingly.

2.1.2 User-controllable Token Denom

Severity High

Status Fixed in Version 2

Introduced by Version 1

Description In the current implementation, users can transfer tokens in the source chain to the contract channel-escrow for cross-chain operations. In this case, function ft_on_transfer() will be invoked to generate the corresponding Ics20TransferRequest for subsequent execution of send_transfer(). However, according to NEP-141, the field token_denom in the Ics20TransferRequest is assigned based on the msg provided by the user.

```
80    /// Callback function for ‘ft_transfer_call‘ of NEP-141 compatible contracts
81    pub fn ft_on_transfer(
```

```
82          &mut self,
83          sender_id: AccountId,
84          amount: U128,
85          msg: String,
86      ) -> PromiseOrValue<U128> {
87          assert!(
88              self.token_contracts
89                  .values()
90                  .into_iter()
91                  .any(|id| env::predecessor_account_id().eq(id)),
92              "ERR_UNREGISTERED_TOKEN_CONTRACT"
93          );
94          assert!(
95              !self.pending_transfer_requests.contains_key(&sender_id),
96              "ERR_PENDING_TRANSFER_REQUEST_EXISTS"
97          );
98          let parse_result: Result<FtOnTransferMsg, _> = serde_json::from_str(msg.as_str());
99          assert!(
100             parse_result.is_ok(),
101             "Invalid msg '{}' attached in 'ft_transfer_call'. Refund deposit.",
102             msg
103         );
104         let msg = parse_result.unwrap();
105         let current_account_id = env::current_account_id();
106         let (channel_id, _) = current_account_id.as_str().split_once(".").unwrap();
107         let transfer_request = Ics20TransferRequest {
108             port_on_a: PORT_ID_STR.to_string(),
109             chan_on_a: channel_id.to_string(),
110             token_trace_path: String::new(),
111             token_denom: msg.token_denom,
112             amount,
113             sender: sender_id.to_string(),
114             receiver: msg.receiver,
115         };
116         ext_transfer_request_handler::ext(self.near_ibc_account())
117             .with_attached_deposit(0)
118             .with_static_gas(utils::GAS_FOR_COMPLEX_FUNCTION_CALL)
119             .with_unused_gas_weight(0)
120             .process_transfer_request(transfer_request.clone());
121         self.pending_transfer_requests
122             .insert(sender_id, transfer_request);
123
124         PromiseOrValue::Value(0.into())
125     }
```

Listing 2.2: channel-escrow/src/lib.rs

Impact Tokens of higher value can be minted on the target chain.

Suggestion Assigning the `token_denom` directly based on the type of tokens transferred in.

2.1.3 Lack of Access Control in register_assets()

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `channel-escrow`, the function `register_asset()` should be a privileged function as it restricts the tokens supported for cross-chain operation. However, no access control is implemented.

```
149 fn register_asset(&mut self, denom: String, token_contract: AccountId) {
150         assert!(
151             !self
152                 .token_contracts
153                 .values()
154                 .into_iter()
155                 .any(|id| id == &token_contract),
156             "ERR_TOKEN_CONTRACT_ALREADY_REGISTERED"
157         );
158         self.token_contracts.insert(denom, token_contract);
159     }
```

Listing 2.3: channel-escrow/src/lib.rs

Impact Any token can be registered on the whitelist of the contract.

Suggestion Implement the logic of access control correspondingly.

2.1.4 Lack of Handling for Failure Transfers

Severity Medium

Status Confirmed

Introduced by `Version 1`

Description The function `do_transfer()` is invoked by the contract to transfer locked tokens to a specific receiver. However, if the receiver is not registered in the token contract, the cross-contract invocation `ft_transfer()` will fail. There is no handling for failed transfers in this function.

The similar issue also exists in the function `cancel_transfer_request()`.

```
161     fn do_transfer(&mut self, base_denom: String, receiver_id: AccountId, amount: U128) {
162         self.assert_near_ibc_account();
163         assert!(
164             self.token_contracts.contains_key(&base_denom),
165             "ERR_INVALID_TOKEN_DENOM"
166         );
167         let token_contract = self.token_contracts.get(&base_denom).unwrap();
168         ext_ft_core::ext(token_contract.clone())
169             .with_attached_deposit(1)
170             .with_static_gas(utils::GAS_FOR_SIMPLE_FUNCTION_CALL)
171             .with_unused_gas_weight(0)
172             .ft_transfer(receiver_id, amount.into(), None);
173     }
```

Listing 2.4: channel-escrow/src/lib.rs

```
187    fn cancel_transfer_request(&mut self, base_denom: String, sender_id: AccountId, amount: U128)
           {
188        self.assert_near_ibc_account();
189        assert!(
190            self.token_contracts.contains_key(&base_denom),
191            "ERR_INVALID_TOKEN_DENOM"
192        );
193        self.checked_remove_pending_transfer_request(&base_denom, &sender_id, amount);
194        let token_contract = self.token_contracts.get(&base_denom).unwrap();
195        ext_ft_core::ext(token_contract.clone())
196            .with_attached_deposit(1)
197            .with_static_gas(utils::GAS_FOR_SIMPLE_FUNCTION_CALL * 2)
198            .with_unused_gas_weight(0)
199            .ft_transfer(sender_id, amount.into(), None);
200    }
```

Listing 2.5: channel-escrow/src/lib.rs

Impact The tokens will be lost.

Suggestion Log an event for the failed transfer to allow the admin to recover lost tokens for users.

Feedback from the Project We'll check for recipient accounts on the `front-end` of our bridge (e.g., connecting to the user's `NEAR` wallet, etc.) and prompt the user that they should make sure that the recipient account is valid, and help them register their recipient account in the token contract. If in the worst case scenario this error occurs, we can manually confirm the specific transaction and process the transfer manually.

2.1.5 Inconsistent Use of Storage Key

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the functions `get_near_ibc_store()` and `set_near_ibc_store()`, the `storage_read()` and `storage_write()` methods are intended to operate on the same struct `NearIbcStore`. However, they use different storage keys. Specifically, the method `storage_read()` uses `StorageKey::NearIbcStore` as the storage key for reading data, while the method `storage_write()` uses `b"ibc_store"` as the storage key for writing data. This inconsistency needs to be addressed.

```
67    fn get_near_ibc_store() -> NearIbcStore {
68        let store =
69            near_sdk::env::storage_read(&StorageKey::NearIbcStore.try_to_vec().unwrap()).unwrap();
70        let store = NearIbcStore::try_from_slice(&store).unwrap();
71        store
72    }
73    ///
74    fn set_near_ibc_store(store: &NearIbcStore) {
```

```
75        let store = store.try_to_vec().unwrap();
76        near_sdk::env::storage_write(b"ibc_store", &store);
77    }
78}
```

Listing 2.6: near-ibc/src/context.rs

Impact Reading and writing data will be executed in different locations in the storage, which leads to unexpected behavior and data inconsistencies.

Suggestion Choose either `StorageKey::NearIbcStore` or `b''ibc_store''` as the storage key for both methods.

2.2 DeFi Security

2.2.1 Improper Check of Attached NEAR Value

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description Function `deliver()` allows the `relayer` to relay messages between chains. The contract performs certain operations based on the input `messages`, which may increase the contract's storage and require storage fees. The `relayer` is responsible for paying the fees. However, the `relayer` can send multiple messages at once (line 127). The current check for the minimum value attached is insufficient and may not cover all the storage fees.

```
126    #[payable]
127    pub fn deliver(&mut self, messages: Vec<Any>) {
128        assert!(
129            env::attached_deposit() >= utils::MINIMUM_DEPOSIT_FOR_DELEVER_MSG,
130            "Need to attach at least {} yocto NEAR to cover the possible storage cost.",
131            utils::MINIMUM_DEPOSIT_FOR_DELEVER_MSG
132        );
133        let used_bytes = env::storage_usage();
134        // Deliver messages to `ibc-rs`
135        let mut near_ibc_store = self.near_ibc_store.get().unwrap();
136
137        let errors = messages.into_iter().fold(vec![], |mut errors, msg| {
138            match MsgEnvelope::try_from(msg) {
139                Ok(msg) => match ibc::core::dispatch(&mut near_ibc_store, msg) {
140                    Ok(()) => (),
141                    Err(e) => errors.push(e),
142                },
143                Err(e) => errors.push(e),
144            }
145            errors
146        });
147        if errors.len() > 0 {
148            log!("Error(s) occurred: {:?}", errors);
149        }
```

```
150         self.near_ibc_store.set(&near_ibc_store);
151
152         // Refund unused deposit.
153         utils::refund_deposit(used_bytes, env::attached_deposit());
154     }
```

Listing 2.7: near-ibc/src/lib.rs

Impact The storage fee in the contract account may be used up, resulting in a denial of service.

Suggestion Attach the value based on the length of the messages.

2.2.2 Attached NEAR Repeatedly Used in Multiple Contracts

Severity Medium

Status Fixed in in `Version 2`

Introduced by `Version 1`

Description As mentioned in `Issue 2.1.4`, function `deliver()` allows the `relayer` to relay multiple messages between chains in one transaction. More than one contract (e.g., `near-ibc`, `<asset id>.tf.transfer.<near ibc>`) requires the storage fee while the `relayer` only pays the storage fee once. The additional fees are paid by the contract itself.

```
126     #[payable]
127     pub fn deliver(&mut self, messages: Vec<Any>) {
128         assert!(
129             env::attached_deposit() >= utils::MINIMUM_DEPOSIT_FOR_DELEVER_MSG,
130             "Need to attach at least {} yocto NEAR to cover the possible storage cost.",
131             utils::MINIMUM_DEPOSIT_FOR_DELEVER_MSG
132         );
133         let used_bytes = env::storage_usage();
134         // Deliver messages to `ibc-rs`
135         let mut near_ibc_store = self.near_ibc_store.get().unwrap();
136
137         let errors = messages.into_iter().fold(vec![], |mut errors, msg| {
138             match MsgEnvelope::try_from(msg) {
139                 Ok(msg) => match ibc::core::dispatch(&mut near_ibc_store, msg) {
140                     Ok(()) => (),
141                     Err(e) => errors.push(e),
142                 },
143                 Err(e) => errors.push(e),
144             }
145             errors
146         });
147         if errors.len() > 0 {
148             log!("Error(s) occurred: {:?}", errors);
149         }
150         self.near_ibc_store.set(&near_ibc_store);
151
152         // Refund unused deposit.
153         utils::refund_deposit(used_bytes, env::attached_deposit());
154     }
```

Listing 2.8: near-ibc/src/lib.rs

```
68  fn mint_coins_execute(
69      &mut self,
70      account: &Self::AccountId,
71      amt: &PrefixedCoin,
72  ) -> Result<(), TokenTransferError> {
73      log!(
74          "Minting coins for account {}, trace path {}, base denom {}",
75          account.0,
76          amt.denom.trace_path,
77          amt.denom.base_denom
78      );
79      ext_token_factory::ext(utils::get_token_factory_contract_id())
80          .with_attached_deposit(env::attached_deposit())
81          .with_static_gas(utils::GAS_FOR_SIMPLE_FUNCTION_CALL * 8)
82          .with_unused_gas_weight(0)
83          .mint_asset(
84              amt.denom.trace_path.to_string(),
85              amt.denom.base_denom.to_string(),
86              account.0.clone(),
87              U128(u128::from_str(amt.amount.to_string().as_str()).unwrap()),
88          );
89      Ok(())
90  }
```

Listing 2.9: near-ibc/ibc_impl/applications/transfer/impls.rs

Impact The initially deposited storage fee in the contract may be used up, resulting in a denial of service.

Suggestion The storage fees for each contract need to be paid by the `relayer`.

2.2.3 Lack of #[payable] Modifier for function register_asset_for_channel()

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The function `register_asset_for_channel()` aims to register token accounts on the whitelist of the contract `channel-escrow`. This operation requires the `governance` account to attach a certain amount of `NEAR` for storage fees. However, there is no `#[payable]` modifier for this function.

```
243  /// Register the given token contract for the given channel.
244  ///
245  /// Only the governance account can call this function.
246  pub fn register_asset_for_channel(
247      &mut self,
248      channel_id: String,
249      denom: String,
250      token_contract: AccountId,
251  ) {
```

```
252         self.assert_governance();
253         let escrow_account_id =
254             format!("{}.{}", channel_id, utils::get_escrow_factory_contract_id());
255         ext_channel_escrow::ext(AccountId::from_str(escrow_account_id.as_str()).unwrap())
256             .with_attached_deposit(env::attached_deposit())
257             .with_static_gas(utils::GAS_FOR_SIMPLE_FUNCTION_CALL)
258             .with_unused_gas_weight(0)
259             .register_asset(denom, token_contract);
260     }
```

Listing 2.10: near-ibc/src/lib.rs

Impact The function `register_asset_for_channel()` cannot receive attached `NEAR`.

Suggestion Add the modifier `#[payable]`.

2.2.4 Incorrect Use of refund_deposit()

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description Function `refund_deposit()` calculates and returns `NEAR` based on the used storage used and the total `NEAR` value attached. However, this function is not correctly used in many places.

Taking the function `deliver()` as an example, the storage used in the cross-contract invocation is not considered in the function `refund_deposit()`. Furthermore, due to the asynchronous feature of `NEAR` protocol, if a relayer modifies the contract's storage state in Block `N`,the function `check_storage_and_refund` may be invoked in `N+M`. In this case, the storage calculation can be influenced by the other transactions between the block `N` and block `N+M`, resulting in incorrect refund storage fee. This issue is also in function `setup_wrapped_token()` and `setup_channel_escrow()`.

```
126     #[payable]
127     pub fn deliver(&mut self, messages: Vec<Any>) {
128         assert!(
129             env::attached_deposit() >= utils::MINIMUM_DEPOSIT_FOR_DELEVER_MSG,
130             "Need to attach at least {} yocto NEAR to cover the possible storage cost.",
131             utils::MINIMUM_DEPOSIT_FOR_DELEVER_MSG
132         );
133         let used_bytes = env::storage_usage();
134         // Deliver messages to `ibc-rs`
135         let mut near_ibc_store = self.near_ibc_store.get().unwrap();
136
137         let errors = messages.into_iter().fold(vec![], |mut errors, msg| {
138             match MsgEnvelope::try_from(msg) {
139                 Ok(msg) => match ibc::core::dispatch(&mut near_ibc_store, msg) {
140                     Ok(()) => (),
141                     Err(e) => errors.push(e),
142                 },
143                 Err(e) => errors.push(e),
144             }
145             errors
146         });
```

```
147         if errors.len() > 0 {
148             log!("Error(s) occurred: {:?}", errors);
149         }
150         self.near_ibc_store.set(&near_ibc_store);
151 
152         // Refund unused deposit.
153         utils::refund_deposit(used_bytes, env::attached_deposit());
154     }
```

Listing 2.11: near-ibc/src/lib.rs

```
163     /// Setup the token contract for the given asset denom with the given metadata.
164     ///
165     /// Only the governance account can call this function.
166     #[payable]
167     pub fn setup_wrapped_token(
168         &mut self,
169         port_id: String,
170         channel_id: String,
171         trace_path: String,
172         base_denom: String,
173         metadata: FungibleTokenMetadata,
174     ) {
175         self.assert_governance();
176         assert!(
177             env::prepaid_gas() >= utils::GAS_FOR_COMPLEX_FUNCTION_CALL,
178             "ERR_NOT_ENOUGH_GAS"
179         );
180         let asset_denom = AssetDenom {
181             trace_path,
182             base_denom,
183         };
184         let minimum_deposit = utils::INIT_BALANCE_FOR_WRAPPED_TOKEN_CONTRACT
185             + env::storage_byte_cost() * (asset_denom.try_to_vec().unwrap().len() + 32) as u128 *
                    2;
186         assert!(
187             env::attached_deposit() >= minimum_deposit,
188             "ERR_NOT_ENOUGH_DEPOSIT, must not less than {} yocto",
189             minimum_deposit
190         );
191         let used_bytes = env::storage_usage();
192         ext_token_factory::ext(utils::get_token_factory_contract_id())
193             .with_attached_deposit(env::attached_deposit())
194             .with_static_gas(
195                 utils::GAS_FOR_COMPLEX_FUNCTION_CALL - utils::GAS_FOR_SIMPLE_FUNCTION_CALL,
196             )
197             .with_unused_gas_weight(0)
198             .setup_asset(
199                 port_id,
200                 channel_id,
201                 asset_denom.trace_path,
202                 asset_denom.base_denom,
203                 metadata,
```

```
            );
        utils::refund_deposit(used_bytes, env::attached_deposit() - minimum_deposit)
    }
```

Listing 2.12: near-ibc/src/lib.rs

```
    /// Setup the escrow contract for the given channel.
    ///
    /// Only the governance account can call this function.
    #[payable]
    pub fn setup_channel_escrow(&mut self, channel_id: String) {
        self.assert_governance();
        assert!(
            env::prepaid_gas() >= utils::GAS_FOR_COMPLEX_FUNCTION_CALL,
            "ERR_NOT_ENOUGH_GAS"
        );
        let minimum_deposit = utils::INIT_BALANCE_FOR_CHANNEL_ESCROW_CONTRACT
            + env::storage_byte_cost() * (channel_id.try_to_vec().unwrap().len() + 16) as u128;
        assert!(
            env::attached_deposit() >= minimum_deposit,
            "ERR_NOT_ENOUGH_DEPOSIT, must not less than {} yocto",
            minimum_deposit
        );
        let used_bytes = env::storage_usage();
        ext_escrow_factory::ext(utils::get_escrow_factory_contract_id())
            .with_attached_deposit(env::attached_deposit())
            .with_static_gas(
                utils::GAS_FOR_COMPLEX_FUNCTION_CALL - utils::GAS_FOR_SIMPLE_FUNCTION_CALL,
            )
            .with_unused_gas_weight(0)
            .create_escrow(ChannelId::from_str(channel_id.as_str()).unwrap());
        utils::refund_deposit(used_bytes, env::attached_deposit() - minimum_deposit);
    }
```

Listing 2.13: near-ibc/src/lib.rs

```
/// Check the usage of storage of current account and refund the unused attached deposit.
///
/// For calling this function, at least `GAS_FOR_CHECK_STORAGE_AND_REFUND` gas is needed.
/// And the contract also needs to call the `impl_storage_check_and_refund!` macro.
///
/// Better to call this function at the end of a `payable` function
/// by recording the `previously_used_bytes` at the start of the `payable` function.
pub fn refund_deposit(previously_used_bytes: u64, max_refundable_amount: u128) {
    #[derive(Serialize, Deserialize, Clone)]
    #[serde(crate = "near_sdk::serde")]
    struct Input {
        pub caller: AccountId,
        pub max_refundable_amount: U128,
        pub previously_used_bytes: U64,
    }
    let args = Input {
        caller: env::predecessor_account_id(),
```

```
58          max_refundable_amount: U128(max_refundable_amount),
59          previously_used_bytes: U64(previously_used_bytes),
60      };
61      let args = near_sdk::serde_json::to_vec(&args)
62          .expect("ERR_SERIALIZE_ARGS_OF_CHECK_STORAGE_AND_REFUND");
63      Promise::new(env::current_account_id()).function_call(
64          "check_storage_and_refund".to_string(),
65          args,
66          0,
67          GAS_FOR_SIMPLE_FUNCTION_CALL,
68      );
69  }
```

Listing 2.14: utils/src/lib.rs

Impact The refunded storage fees will always be less than the actual storage consumption if the function `deliver()` is invoked in subsequent blocks.

Suggestion Avoid cross-contract calls in function `refund_deposit()`.

2.2.5 Lack of Implementation of Storage Check and Refund for register_asset()

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In function `register_asset()`, new token contracts are added to the whitelist in the contract `channel-escrow`, thereby increasing the storage usage of the contract. However, the function does not require the caller to attach `NEAR` for the storage cost, and it does not implement the logic to refund additional `NEAR`.

The similar problem also exists in the function `register_asset_for_channel()`.

```
149     fn register_asset(&mut self, denom: String, token_contract: AccountId) {
150         assert!(
151             !self
152                 .token_contracts
153                 .values()
154                 .into_iter()
155                 .any(|id| id == &token_contract),
156             "ERR_TOKEN_CONTRACT_ALREADY_REGISTERED"
157         );
158         self.token_contracts.insert(denom, token_contract);
159     }
```

Listing 2.15: channel-escrow/src/lib.rs

```
243     /// Register the given token contract for the given channel.
244     ///
245     /// Only the governance account can call this function.
246     pub fn register_asset_for_channel(
247         &mut self,
248         channel_id: String,
```

```
249         denom: String,
250         token_contract: AccountId,
251     ) {
252         self.assert_governance();
253         let escrow_account_id =
254             format!("{}.{}", channel_id, utils::get_escrow_factory_contract_id());
255         ext_channel_escrow::ext(AccountId::from_str(escrow_account_id.as_str()).unwrap())
256             .with_attached_deposit(env::attached_deposit())
257             .with_static_gas(utils::GAS_FOR_SIMPLE_FUNCTION_CALL)
258             .with_unused_gas_weight(0)
259             .register_asset(denom, token_contract);
260     }
```

Listing 2.16: near-ibc/src/lib.rs

Impact According to the design, to register a new asset, the privileged governance account will invoke the function `register_asset_for_channel()`, and then invoke the function `register_asset()` by attaching `NEAR`. In this case, the attached `NEAR` will not be handled.

Suggestion Implement the logic of storage check and refund.

2.2.6 Incorrect Results Returned during Executions

Severity Medium

Status Confirmed

Introduced by `Version 1`

Description The function `mint_coins_execute()` aims to mint tokens for the specified receiver in the token contract created by `token-factory`. Since this operation involves a cross-contract call, it does not directly revert if the execution fails. In this case, the function simply returns the result `Ok(())`, ignoring the possibility of a failure during execution.

The same issue also exists in the function `send_coins_execute()` and `burn_coins_execute()`.

```
35     fn send_coins_execute(
36         &mut self,
37         from: &Self::AccountId,
38         to: &Self::AccountId,
39         amt: &PrefixedCoin,
40     ) -> Result<(), TokenTransferError> {
41         let sender_id = from.0.to_string();
42         let receiver_id = to.0.to_string();
43         let base_denom = amt.denom.base_denom.to_string();
44         if receiver_id.ends_with(env::current_account_id().as_str()) {
45             ext_process_transfer_request_callback::ext(to.0.clone())
46                 .with_attached_deposit(0)
47                 .with_static_gas(utils::GAS_FOR_SIMPLE_FUNCTION_CALL)
48                 .with_unused_gas_weight(0)
49                 .apply_transfer_request(
50                     base_denom,
51                     from.0.clone(),
52                     U128(u128::from_str(amt.amount.to_string().as_str()).unwrap()),
53                 );
```

```
54          } else if sender_id.ends_with(env::current_account_id().as_str()) {
55              ext_channel_escrow::ext(from.0.clone())
56                  .with_attached_deposit(1)
57                  .with_static_gas(utils::GAS_FOR_SIMPLE_FUNCTION_CALL)
58                  .with_unused_gas_weight(0)
59                  .do_transfer(
60                      base_denom,
61                      to.0.clone(),
62                      U128(u128::from_str(amt.amount.to_string().as_str()).unwrap()),
63                  );
64          }
65          Ok(())
66      }
67
68      fn mint_coins_execute(
69          &mut self,
70          account: &Self::AccountId,
71          amt: &PrefixedCoin,
72      ) -> Result<(), TokenTransferError> {
73          log!(
74              "Minting coins for account {}, trace path {}, base denom {}",
75              account.0,
76              amt.denom.trace_path,
77              amt.denom.base_denom
78          );
79          ext_token_factory::ext(utils::get_token_factory_contract_id())
80              .with_attached_deposit(env::attached_deposit())
81              .with_static_gas(utils::GAS_FOR_SIMPLE_FUNCTION_CALL * 8)
82              .with_unused_gas_weight(0)
83              .mint_asset(
84                  amt.denom.trace_path.to_string(),
85                  amt.denom.base_denom.to_string(),
86                  account.0.clone(),
87                  U128(u128::from_str(amt.amount.to_string().as_str()).unwrap()),
88              );
89          Ok(())
90      }
91
92      fn burn_coins_execute(
93          &mut self,
94          account: &Self::AccountId,
95          amt: &PrefixedCoin,
96      ) -> Result<(), TokenTransferError> {
97          log!(
98              "Burning coins for account {}, trace path {}, base denom {}",
99              account.0,
100             amt.denom.trace_path,
101             amt.denom.base_denom
102         );
103         ext_process_transfer_request_callback::ext(env::predecessor_account_id())
104             .with_attached_deposit(0)
105             .with_static_gas(utils::GAS_FOR_SIMPLE_FUNCTION_CALL)
106             .with_unused_gas_weight(0)
```

```
107             .apply_transfer_request(
108                 amt.denom.base_denom.to_string(),
109                 account.0.clone(),
110                 U128(u128::from_str(amt.amount.to_string().as_str()).unwrap()),
111             );
112         Ok(())
113     }
```

Listing 2.17: near-ibc/ibc_impl/applications/transfer/impls.rs

Impact The source chain will not be acknowledged of the failure in the target chain, which results in a loss of assets.

Suggestion Implement the related logic to handle the failure information correctly.

Feedback from the Project As the final results of these 3 functions can not be known before the cross-contract function calls end, we prefer current implementation. Which is, if the following execution failed, we'll handle it manually.

2.2.7 Improper Check in send_coins_execute()

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the function `send_coins_execute()`, the contract checks the `receiver_id` and `sender_id` to determine the transfer actions. When the `receiver_id` ends with the current account id, users are expected to send tokens to the contract `channel-escrow` for transferring the tokens to the target chain. On the contrary, users aim to redeem tokens from the source chain, expecting the contract `channel-escrow` to transfer the locked tokens back.

However, the improper check using `ends_with()`, which is mentioned in `Issue 2.1.1`, may violate the designated execution logic. For example, if the `sender_id` is `<channel id>.ef.transfer.<near ibc>` while `receiver_id` is `foo_<near_ibc>`, the expected execution logic is to transfer the token from `escrow` to the receiver while the actual execution logic is to transfer the tokens to `escrow`.

```
35     fn send_coins_execute(
36         &mut self,
37         from: &Self::AccountId,
38         to: &Self::AccountId,
39         amt: &PrefixedCoin,
40     ) -> Result<(), TokenTransferError> {
41         let sender_id = from.0.to_string();
42         let receiver_id = to.0.to_string();
43         let base_denom = amt.denom.base_denom.to_string();
44         if receiver_id.ends_with(env::current_account_id().as_str()) {
45             ext_process_transfer_request_callback::ext(to.0.clone())
46                 .with_attached_deposit(0)
47                 .with_static_gas(utils::GAS_FOR_SIMPLE_FUNCTION_CALL)
48                 .with_unused_gas_weight(0)
49                 .apply_transfer_request(
50                     base_denom,
```

```
51                 from.0.clone(),
52                 U128(u128::from_str(amt.amount.to_string().as_str()).unwrap()),
53             );
54     } else if sender_id.ends_with(env::current_account_id().as_str()) {
55         ext_channel_escrow::ext(from.0.clone())
56             .with_attached_deposit(1)
57             .with_static_gas(utils::GAS_FOR_SIMPLE_FUNCTION_CALL)
58             .with_unused_gas_weight(0)
59             .do_transfer(
60                 base_denom,
61                 to.0.clone(),
62                 U128(u128::from_str(amt.amount.to_string().as_str()).unwrap()),
63             );
64     }
65     Ok(())
66 }
```

Listing 2.18: near-ibc/ibc_impl/applications/transfer/impls.rs

Impact The execution logic can be wrong in corner cases.

Suggestion Revise the logic accordingly.

2.3 Additional Recommendation

2.3.1 Redundant Uninitialized Check

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The function `new()` is used to initialize the contract. It should not be executed when the contract's state already exists. However, this is already guaranteed by the modifier `#[init]`. Therefore, the assertion of `!env::state_exists()` is redundant.

```
65   #[init]
66   pub fn new(near_ibc_account: AccountId) -> Self {
67       assert!(!env::state_exists(), "ERR_ALREADY_INITIALIZED");
68       let account_id = String::from(env::current_account_id().as_str());
69       let parts = account_id.split(".").collect::<Vec<&str>>();
70       assert!(
71           parts.len() > 2,
72           "ERR_CONTRACT_MUST_BE_DEPLOYED_IN_SUB_ACCOUNT",
73       );
74       Self {
75           near_ibc_account,
76           token_contracts: UnorderedMap::new(StorageKey::TokenContracts),
77           pending_transfer_requests: UnorderedMap::new(StorageKey::PendingTransferRequests),
78       }
79   }
```

Listing 2.19: channel-escrow/src/lib.rs

```rust
#[init]
pub fn new(
    metadata: FungibleTokenMetadata,
    port_id: String,
    channel_id: String,
    trace_path: String,
    base_denom: String,
    near_ibc_account: AccountId,
) -> Self {
    assert!(!env::state_exists(), "ERR_ALREADY_INITIALIZED");
    let account_id = String::from(env::current_account_id().as_str());
    let parts = account_id.split(".").collect::<Vec<&str>>();
    assert!(
        parts.len() > 3,
        "ERR_CONTRACT_MUST_BE_DEPLOYED_IN_SUB_ACCOUNT_OF_FACTORY",
    );
    metadata.assert_valid();
    assert!(
        env::current_account_id()
            .to_string()
            .ends_with(near_ibc_account.as_str()),
        "ERR_NEAR_IBC_ACCOUNT_MUST_HAVE_THE_SAME_ROOT_ACOUNT_AS_CURRENT_ACCOUNT"
    );
    let mut this = Self {
        token: FungibleToken::new(StorageKey::Token),
        metadata: LazyOption::new(StorageKey::Metadata, Some(&metadata)),
        port_id,
        channel_id,
        trace_path,
        base_denom,
        near_ibc_account,
        pending_transfer_requests: UnorderedMap::new(StorageKey::PendingBurnings),
    };
    this.token
        .internal_register_account(&env::current_account_id());
    this
}
```

Listing 2.20: wrapped-token/src/lib.rs

```rust
#[init]
pub fn new() -> Self {
    assert!(!env::state_exists(), "ERR_ALREADY_INITIALIZED");
    let account_id = String::from(env::current_account_id().as_str());
    let parts = account_id.split(".").collect::<Vec<&str>>();
    assert!(
        parts.len() > 2,
        "ERR_CONTRACT_MUST_BE_DEPLOYED_IN_SUB_ACCOUNT",
    );
    Self {
        asset_id_mappings: UnorderedMap::new(StorageKey::AssetIdMappings),
        denom_mappings: UnorderedMap::new(StorageKey::DenomMappings),
    }
```

```
45    }
```

Listing 2.21: token-factory/src/lib.rs

```
45    #[init]
46    pub fn new() -> Self {
47        assert!(!env::state_exists(), "ERR_ALREADY_INITIALIZED");
48        let account_id = String::from(env::current_account_id().as_str());
49        let parts = account_id.split(".").collect::<Vec<&str>>();
50        assert!(
51            parts.len() > 2,
52            "ERR_CONTRACT_MUST_BE_DEPLOYED_IN_SUB_ACCOUNT",
53        );
54        Self {
55            channel_id_set: UnorderedSet::new(StorageKey::ChannelIdSet),
56        }
57    }
```

Listing 2.22: escrow-factory/src/lib.rs

Suggestion I Remove the assertion of `!env::state_exists()`.

2.3.2 Lack of Check on denom

Status Confirmed

Introduced by `Version 1`

Description In function `register_asset()`, a token contract is added to the whitelist in the contract by inserting corresponding key (i.e., `denom`) and value (i.e., `token_contract`) to `token_contracts`. During registration, it is suggested to check whether the characters of `denom` are legal.

```
149   fn register_asset(&mut self, denom: String, token_contract: AccountId) {
150       assert!(
151           !self
152               .token_contracts
153               .values()
154               .into_iter()
155               .any(|id| id == &token_contract),
156           "ERR_TOKEN_CONTRACT_ALREADY_REGISTERED"
157       );
158       self.token_contracts.insert(denom, token_contract);
159   }
```

Listing 2.23: channel-escrow/src/lib.rs

Suggestion I Add the check to ensure the characters of `denom` are legal.

Feedback from the Project The original calling of this function is from the `governance` account of the bridge. The `denom` should have already be checked manually.

2.4 Notes

2.4.1 Assumption on the Secure Implementation of Dependencies

Status Confirmed

Introduced by `version 1`

Description The `Near-IBC` protocol is built based on the crates `near-sdk`(version 4.1.0), `near-contract-standards`(version 4.1.0), and `ibc-rs`(version 0.42.0). In this audit, we assume the standard library provided by `NEAR-SDK-RS` (i.e., `near_contract_standards`) and `IBC-RS` have no security issues.

2.4.2 Storage Consumption when Relay Messages

Status Confirmed

Introduced by `version 1`

Description `Relayers` have to pay the storage fees incurred by the contract during the execution of messages through the function `deliver()`, which facilitates the transmission of valid messages between chains.

2.4.3 Source Chain Unavailable for Multiple Cross Chain Redemption

Status Confirmed

Introduced by `version 1`

Description The source chain tokens do not have the capability to be redeemed after being transferred from the source chain to chain `B` and then further transferred to chain `C`. To transfer back to the original chain, it's necessary to retrace the same path (i.e., `C` -> `B` -> `source chain`) taken during the cross-chain transfer.

2.4.4 Failed Retrieval of Host Consensus State

Status Confirmed

Introduced by `version 1`

Description In the current implementation, the details of the consensus state of the host blockchain can not be validated in the smart contract. Therefore, when the contract calls the relevant check interface in `ibc-rs`, it does not execute the corresponding check logic. For example, in the function `validate_self_client()`, the contract will always return `OK(())`.