Security Audit Report for Octopus-Appchain-Anchor

Date: July 10th, 2022

Version: 1.0

Contact: contact@blocksec.com

BlockSec Audit Team

Digitally signed by BlockSec Audit Team
Date: 2022.07.13 16:28:14 +08'00'

Contents

1 Introduction **1**

1.1 About Target Contracts . . . 1
1.2 Disclaimer . . . 2
1.3 Procedure of Auditing . . . 2
1.3.1 Software Security . . . 3
1.3.2 DeFi Security . . . 3
1.3.3 NFT Security . . . 3
1.3.4 Additional Recommendation . . . 3
1.4 Security Model . . . 4

2 Findings **5**

2.1 Software Security . . . 5
2.1.1 Case-Sensitive Problem . . . 5
2.1.2 Potential DoS Problem . . . 6
2.1.3 Assertion Missing for Wrapped Appchain NFT Registration . . . 7
2.1.4 Validation Missing for Deserialized LockNftPayload.receiver_id . . . 8
2.2 DeFi Security . . . 10
2.2.1 Potential over Unlocked Near Fungible Token . . . 10
2.2.2 Token Price may be Outdated . . . 12
2.2.3 Callback Functions are Missing . . . 13
2.2.4 Tokens may be Locked When Bridging is Closed . . . 15
2.2.5 Incorrect Wrapped Appchain Token Management . . . 16
2.3 Additional Recommendation . . . 19
2.3.1 Redundant Code . . . 19
2.3.2 Assertion Missing for Changing NEAR Fungible Token Metadata . . . 20
2.3.3 Potential Precision Loss . . . 21
2.3.4 Potential Unreasonable Protocol Settings . . . 21
2.3.5 Potential Inconsistencies between Code and Storage . . . 22
2.3.6 Missing Check on the Range of the Validator Count . . . 23
2.3.7 Potential Centralization Problem . . . 24
2.3.8 Potential Elastic Supply Token Problem . . . 24
2.4 Additional Note . . . 25
2.4.1 Errors from the Appchain are Ignored . . . 25

Report Manifest

Item	Description
Client	Octopus Network
Target	Octopus-Appchain-Anchor

Version History

Version	Date	Description
1.0	July 10th, 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 Octopus-Appchain-Anchor [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 code (in the following versions) to fix issues in the audit report.

Project	Version	Commit SHA
Octopus-Appchain-Anchor	`Version 1`	`9b7ed50b9fda8d45c0dfba38a03222a3fc8dc9aa`
	`Version 2`	`a3ee2698ff70123f1776b50691654c2e15821c29`

Note that we did **NOT** audit all the code in the repository. The scope of this audit report **ONLY** include the following files under the directory **appchain-anchor/src**.

- lib.rs
- interfaces.rs
- anchor_viewer.rs
- validator_profiles.rs
- reward_distribution_records.rs
- lookup_array.rs
- message_decoder.rs
- storage_key.rs
- types.rs
- storage_migration.rs
- upgrade.rs
- appchain_messages.rs
- user_staking_histories.rs
- appchain_challenge/
  - equivocation_challenge.rs
  - mod.rs
- assets/
  - mod.rs
  - near_fungible_tokens.rs
  - wrapped_appchain_nfts.rs
  - wrapped_appchain_token.rs

[1] https://github.com/octopus-network/octopus-appchain-anchor

- permissionless_actions/
  - distributing_rewards.rs
  - mod.rs
  - switching_era.rs
- user_actions/
  - appchain_lifecycle.rs
  - mod.rs
  - owner_actions.rs
  - settings_manager.rs
  - staking.rs
  - sudo_actions.rs
  - validator_actions.rs
- validator_set/
  - mod.rs
  - next_validator_set.rs
  - validator_set_of_era.rs

All the code in the other files, which are not mentioned above, of this repository is out of our audit scope.

1.2 Disclaimer

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

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

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

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **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 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.

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

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

Chapter 2 Findings

In total, we find 9 potential issues in the smart contract. We also have 8 recommendations and 1 note, as follows:

- Medium Risk: 5
- Low Risk: 4
- Recommendations: 8
- Notes: 1

ID	Severity	Description	Category	Status
1	Medium	*Case-Sensitive Problem*	Software Security	Fixed
2	Medium	*Potential DoS Problem*	Software Security	Confirmed
3	Medium	*Assertion Missing for Wrapped Appchain NFT Registration*	Software Security	Fixed
4	Low	*Validation Missing for Deserialized LockNftPayload.receiver_id*	Software Security	Fixed
5	Medium	*Potential over Unlocked Near Fungible Token*	DeFi Security	Fixed
6	Low	*Token Price may be Outdated*	DeFi Security	Confirmed
7	Low	*Callback Functions are Missing*	DeFi Security	Confirmed
8	Medium	*Tokens may be Locked When Bridging is Closed*	DeFi Security	Fixed
9	Low	*Incorrect Wrapped Appchain Token Management*	DeFi Security	Confirmed
11	-	*Redundant Code*	Recommendation	Fixed
12	-	*Assertion Missing for Changing NEAR Fungible Token Metadata*	Recommendation	Fixed
13	-	*Potential Precision Loss*	Recommendation	Confirmed
14	-	*Potential Unreasonable Protocol Settings*	Recommendation	Fixed
15	-	*Potential Inconsistencies between Code and Storage*	Recommendation	Fixed
16	-	*Missing Check on the Range of the Validator Count*	Recommendation	Fixed
17	-	*Potential Centralization Problem*	Recommendation	Confirmed
18	-	*Potential Elastic Supply Token Problem*	Recommendation	Confirmed
19	-	*Errors from the Appchain are Ignored*	Note	Confirmed

The details are provided in the following sections.

2.1 Software Security

2.1.1 Case-Sensitive Problem

Status Fixed in `version 2`

Introduced by `version 1`

Description The `id_in_appchain` in function `AccountIdInAppchain::new()` is case-sensitive. Meanwhile, the real ID in appchain is case-insensitive. In this case, two different account IDs can map to the same ID in the appchain.

```
15    impl AccountIdInAppchain {
16        ///
17        pub fn new(id_in_appchain: Option<String>) -> Self {
18            let mut value = String::new();
19            if let Some(id_in_appchain) = id_in_appchain.clone() {
```

```
20              if !id_in_appchain.starts_with("0x") {
21                  value.push_str("0x");
22              }
23              value.push_str(&id_in_appchain);
24          }
25          Self {
26              origin: id_in_appchain,
27              raw_string: value,
28          }
29      }
```

Listing 2.1: appchain-anchor/src/types.rs

Impact All the assertions associated with the `AccountIdInAppchain` may be failed and can lead to high security impacts (e.g., reward cannot be claimed successfully).

Suggestion I Implement function `AccountIdInAppchain::new` as a case-insensitive one.

2.1.2 Potential DoS Problem

Status Confirmed

Introduced by `version 1`

Description The maximum prepaid gas (300T) allowed for one transaction may still not be enough for function `go_booting` invocation. In this case, the process of `complete_switching_era` (line 35) should be split into several transactions. However, the loop (lines 34-46) won't break if the maximum prepaid gas is exceeded.

```
 9  fn go_booting(&mut self) {
10      self.assert_owner();
11      assert_eq!(
12          self.appchain_state,
13          AppchainState::Staging,
14          "Appchain state must be 'staging'."
15      );
16      let protocol_settings = self.protocol_settings.get().unwrap();
17      let next_validator_set = self.next_validator_set.get().unwrap();
18      assert!(
19          next_validator_set.validator_count() >= protocol_settings.minimum_validator_count.0,
20          "Not enough validators available."
21      );
22      let oct_token = self.oct_token.get().unwrap();
23      assert!(
24          next_validator_set.total_stake() / OCT_DECIMALS_VALUE * oct_token.price_in_usd.0
25              >= protocol_settings.minimum_total_stake_price_for_booting.0,
26          "Not enough stake deposited in anchor."
27      );
28      self.appchain_state = AppchainState::Booting;
29      let mut processing_context = AppchainMessagesProcessingContext::new(
30          self.permissionless_actions_status.get().unwrap(),
31      );
32      let mut validator_set_histories = self.validator_set_histories.get().unwrap();
33      self.internal_start_switching_era(&mut processing_context, &mut validator_set_histories, 0)
            ;
```

```
34        loop {
35            match self.complete_switching_era(
36                &mut processing_context,
37                &mut validator_set_histories,
38                0,
39            ) {
40                MultiTxsOperationProcessingResult::Ok => break,
41                MultiTxsOperationProcessingResult::NeedMoreGas => (),
42                MultiTxsOperationProcessingResult::Error(message) => {
43                    panic!("Failed to generate validator set 0: '{}'", &message)
44                }
45            }
46        }
47        self.validator_set_histories.set(&validator_set_histories);
48        self.sync_state_to_registry();
49    }
```

Listing 2.2: appchain-anchor/src/user_actions/appchain_lifecycle.rs

Impact The contract owner may not be able to boot the contract up permanently.

Suggestion I It is suggested to change the `appchain_state` to be `AppchainState::Booting` only when the era switching process is done entirely in line 40. Besides, it is also suggested to break the loop in the case of `NeedMoreGas` to prevent an infinite loop in line 41.

Feedback from the Project In current process, the booting is controlled by the admin, and the first `validator_set` will only have 4 validators. So the gas consumption of this function will not exceed 200T actually. I think we can keep current implementation (take the risk).

2.1.3 Assertion Missing for Wrapped Appchain NFT Registration

Status Fixed in `version 2`

Introduced by `version 1`

Description The existence of the NFT token for `class_id` is not checked in function `register_wrapped_appchain_nft`.

```
156    fn register_wrapped_appchain_nft(&mut self, class_id: String, metadata: NFTContractMetadata) {
157        self.assert_owner();
158        assert!(
159            env::storage_has_key(&StorageKey::WrappedAppchainNFTContractWasm.into_bytes()),
160            "Wasm file for deployment is not staged yet."
161        );
162        let internal_wrapped_appchain_nft =
163            InternalWrappedAppchainNFT::new(class_id.clone(), metadata.clone());
164        let mut wrapped_appchain_nfts = self.wrapped_appchain_nfts.get().unwrap();
165        wrapped_appchain_nfts.insert(&class_id, &internal_wrapped_appchain_nft);
166        self.wrapped_appchain_nfts.set(&wrapped_appchain_nfts);
167        //
168        #[derive(near_sdk::serde::Serialize)]
169        #[serde(crate = "near_sdk::serde")]
170        struct Input {
171            owner_id: AccountId,
172            metadata: NFTContractMetadata,
```

```
173         }
174         let args = Input {
175             owner_id: env::current_account_id(),
176             metadata,
177         };
178         let args = near_sdk::serde_json::to_vec(&args)
179             .expect("Failed to serialize the cross contract args using JSON.");
180         Promise::new(internal_wrapped_appchain_nft.contract_account)
181             .create_account()
182             .transfer(WRAPPED_APPCHAIN_NFT_CONTRACT_INIT_BALANCE)
183             .add_full_access_key(self.owner_pk.clone())
184             .deploy_contract(
185                 env::storage_read(&StorageKey::WrappedAppchainNFTContractWasm.into_bytes())
186                     .unwrap(),
187             )
188             .function_call(
189                 "new".to_string(),
190                 args,
191                 0,
192                 Gas::ONE_TERA.mul(T_GAS_FOR_NFT_CONTRACT_INITIALIZATION),
193             );
194     }
```

Listing 2.3: appchain-anchor/src/assets/wrapped_appchain_nfts.rs

Impact Wrapped appchain NFT tokens may be re-initialized.

Suggestion I Check the existence of the NFT token for `class_id` in function `register_wrapped_appchain_nft`.

2.1.4 Validation Missing for Deserialized LockNftPayload.receiver_id

Status Fixed in `version 2`

Introduced by `version 1`

Description The `MessagePayload` for `LockNftPayload.receiver_id` is deserialized into type `String` instead of `AccountId` in function `decode` (line 153-154).

```
44 #[derive(BorshSerialize, BorshDeserialize, Serialize, Deserialize, Clone)]
45 #[serde(crate = "near_sdk::serde")]
46 pub struct LockNftPayload {
47     pub sender: String,
48     pub receiver_id: String,
49     pub class: u128,
50     pub instance: u128,
51     pub metadata: TokenMetadata,
52 }
```

Listing 2.4: appchain-anchor/src/message_decoder.rs

```
151     // Code snippet of function decode:
152     PayloadType::LockNft => {
153         let payload_result: Result<LockNftPayload, std::io::Error> =
154             BorshDeserialize::deserialize(&mut &m.payload[..]);
```

```
155         let payload = payload_result.unwrap();
156         log!(
157             "Origin appchain message: '{}'",
158             serde_json::to_string(&payload).unwrap()
159         );
160         AppchainMessage {
161             nonce: m.nonce as u32,
162             appchain_event: AppchainEvent::NonFungibleTokenLocked {
163                 owner_id_in_appchain: payload.sender,
164                 receiver_id_in_near: AccountId::new_unchecked(payload.receiver_id),
165                 class_id: payload.class.to_string(),
166                 instance_id: payload.instance.to_string(),
167                 token_metadata: payload.metadata,
168             },
169         }
170     }
```

Listing 2.5: appchain-anchor/src/message_decoder.rs

Thus, function `validate_account_id` won't be invoked when deserializing the raw string into the `LockNftPayload.receiver_id`.

```
 95     fn validate_account_id(id: &str) -> Result<(), ParseAccountIdError> {
 96         if is_valid_account_id(id.as_bytes()) {
 97             Ok(())
 98         } else {
 99             Err(ParseAccountIdError {})
100         }
101     }
102
103     impl TryFrom<String> for AccountId {
104         type Error = ParseAccountIdError;
105
106         fn try_from(value: String) -> Result<Self, Self::Error> {
107             validate_account_id(value.as_str())?;
108             Ok(Self(value))
109         }
110     }
111
112     impl std::str::FromStr for AccountId {
113         type Err = ParseAccountIdError;
114
115         fn from_str(value: &str) -> Result<Self, Self::Err> {
116             validate_account_id(value)?;
117             Ok(Self(value.to_string()))
118         }
119     }
```

Listing 2.6: near-sdk-4.0.0/src/types/account_id.rs

Impact AppchainAnchor may take an invalid `AccountId` when constructing the `AppchainMessage` for `PayloadType::LockNft`.

Suggestion I Define the attribute `receiver_id` of `LockNftPayload` as the type of `AccountId` rather than

the `String`.

2.2 DeFi Security

2.2.1 Potential over Unlocked Near Fungible Token

Status Fixed in `version 2`

Introduced by `version 1`

Description The `NearFungibleToken.locked_balance` is used to maintain the total balance of the near-fungible token locked in this contract. However, when unlocking the near-fungible token from this contract by invoking the function `internal_unlock_near_fungible_token`, the `NearFungibleToken.locked_balance` is updated in its callback function `resolve_fungible_token_transfer` (line 352-358 of listing 2.8).

If the total balance of the unlocking near-fungible tokens for several `AppchainEvents::NearFungibleTokenBurnt` applied in the same block is greater than the `NearFungibleToken.locked_balance`, the number of tokens transferred may exceed the limitation.

```
271     pub fn internal_unlock_near_fungible_token(
272         &mut self,
273         sender_id_in_appchain: String,
274         contract_account: AccountId,
275         receiver_id_in_near: AccountId,
276         amount: U128,
277         appchain_message_nonce: u32,
278         processing_context: &mut AppchainMessagesProcessingContext,
279     ) -> MultiTxsOperationProcessingResult {
280         let near_fungible_tokens = self.near_fungible_tokens.get().unwrap();
281         if let Some(near_fungible_token) =
282             near_fungible_tokens.get_by_contract_account(&contract_account)
283         {
284             if near_fungible_token
285                 .bridging_state
286                 .eq(&BridgingState::Closed)
287             {
288                 let message = format!(
289                     "Bridging for NEAR fungible token in contract '{}' is now closed.",
290                     contract_account
291                 );
292                 let result = AppchainMessageProcessingResult::Error {
293                     nonce: appchain_message_nonce,
294                     message: message.clone(),
295                 };
296                 self.record_appchain_message_processing_result(&result);
297                 return MultiTxsOperationProcessingResult::Error(message);
298             }
299             ext_ft_core::ext(near_fungible_token.contract_account)
300                 .with_attached_deposit(1)
301                 .with_static_gas(Gas::ONE_TERA.mul(T_GAS_FOR_FT_TRANSFER))
302                 .with_unused_gas_weight(0)
303                 .ft_transfer(receiver_id_in_near.clone(), amount, None)
304                 .then(
```

```
                ext_self::ext(env::current_account_id())
                    .with_attached_deposit(0)
                    .with_static_gas(Gas::ONE_TERA.mul(T_GAS_FOR_RESOLVER_FUNCTION))
                    .with_unused_gas_weight(0)
                    .resolve_fungible_token_transfer(
                        near_fungible_token.metadata.symbol,
                        sender_id_in_appchain,
                        receiver_id_in_near.clone(),
                        amount,
                        appchain_message_nonce,
                    ),
            );
        processing_context.add_prepaid_gas(Gas::ONE_TERA.mul(T_GAS_FOR_FT_TRANSFER));
        processing_context.add_prepaid_gas(Gas::ONE_TERA.mul(T_GAS_FOR_RESOLVER_FUNCTION));
        MultiTxsOperationProcessingResult::Ok
```

Listing 2.7: appchain-anchor/src/assets/near_fungible_tokens.rs

The callback function `resolve_fungible_token_transfer`:

```
#[near_bindgen]
impl FungibleTokenContractResolver for AppchainAnchor {
    //
    fn resolve_fungible_token_transfer(
        &mut self,
        symbol: String,
        sender_id_in_appchain: String,
        receiver_id_in_near: AccountId,
        amount: U128,
        appchain_message_nonce: u32,
    ) {
        assert_self();
        match env::promise_result(0) {
            PromiseResult::NotReady => unreachable!(),
            PromiseResult::Successful(_) => {
                let mut near_fungible_tokens = self.near_fungible_tokens.get().unwrap();
                if let Some(mut near_fungible_token) = near_fungible_tokens.get(&symbol) {
                    near_fungible_token.locked_balance =
                        match near_fungible_token.locked_balance.0.checked_sub(amount.0) {
                            Some(value) => U128::from(value),
                            None => U128::from(0),
                        };
                    near_fungible_tokens.insert(&near_fungible_token);
                };
                self.record_appchain_message_processing_result(
                    &AppchainMessageProcessingResult::Ok {
                        nonce: appchain_message_nonce,
                        message: None,
                    },
                );
            }
            PromiseResult::Failed => {
                let reason = format!(
                    "Maybe the receiver account '{}' is not registered in '{}' token contract.",
```

```
369                 &receiver_id_in_near, &symbol
370             );
371             let message = format!(
372                 "Failed to unlock near fungible token for appchain account '{}'. {}",
373                 sender_id_in_appchain, reason
374             );
375             self.record_appchain_message_processing_result(
376                 &AppchainMessageProcessingResult::Error {
377                     nonce: appchain_message_nonce,
378                     message,
379                 },
380             );
381         }
382     }
383 }
384}
```

Listing 2.8: appchain-anchor/src/assets/near_fungible_tokens.rs

Impact The total amount of near-fungible token unlocked may excced the limitation of its `NearFungibleToken.locked_balance`.

Suggestion I Reduce the `NearFungibleToken.locked_balance` before applying the cross-contract invocation `ft_transfer`.

2.2.2 Token Price may be Outdated

Status Confirmed

Introduced by `version 1`

Description When setting the prices of `oct_token`, `near_fungible_token`, and the `wrapped_appchain_token`, one certain extra attribute `last_updated_time = env::block_timestamp()` is not maintained for each token. Given this, the freshness of the token prices cannot be checked.

```
440     /// Set the price (in USD) of OCT token
441     pub fn set_price_of_oct_token(&mut self, price: U128) {
442         self.assert_token_price_maintainer();
443         let mut oct_token = self.oct_token.get().unwrap();
444         oct_token.price_in_usd = price;
445         self.oct_token.set(&oct_token);
446     }
```

Listing 2.9: appchain-anchor/src/lib.rs

```
157     fn set_price_of_near_fungible_token(&mut self, symbol: String, price: U128) {
158         self.assert_token_price_maintainer();
159         let mut near_fungible_tokens = self.near_fungible_tokens.get().unwrap();
160         assert!(
161             near_fungible_tokens.contains(&symbol),
162             "Token '{}' is not registered.",
163             &symbol
164         );
165         let mut near_fungible_token = near_fungible_tokens.get(&symbol).unwrap();
```

```
166         near_fungible_token.price_in_usd = price;
167         near_fungible_tokens.insert(&near_fungible_token);
168     }
```

Listing 2.10: appchain-anchor/src/assets/near_fungible_tokens.rs

```
103     fn set_price_of_wrapped_appchain_token(&mut self, price: U128) {
104         self.assert_token_price_maintainer();
105         let mut wrapped_appchain_token = self.wrapped_appchain_token.get().unwrap();
106         wrapped_appchain_token.price_in_usd = price;
107         self.wrapped_appchain_token.set(&wrapped_appchain_token);
108     }
```

Listing 2.11: appchain-anchor/src/assets/wrapped_appchain_token.rs

Impact The values calculated by the functions associated with the token prices may be outdated (e.g., the calculation of the market value).

```
71 pub fn total_market_value(&self) -> Balance {
72     let mut total_market_value: u128 = 0;
73     let symbols = self.symbols.to_vec();
74     symbols.iter().for_each(|symbol| {
75         let near_fungible_token = self.tokens.get(&symbol).unwrap();
76         total_market_value += near_fungible_token.locked_balance.0
77             / u128::pow(10, u32::from(near_fungible_token.metadata.decimals))
78             * near_fungible_token.price_in_usd.0
79     });
80     total_market_value
81 }
82 ///
83 pub fn get_market_value_of(&self, symbol: &String, amount: u128) -> Balance {
84     if let Some(near_fungible_token) = self.tokens.get(&symbol) {
85         amount / u128::pow(10, u32::from(near_fungible_token.metadata.decimals))
86             * near_fungible_token.price_in_usd.0
87     } else {
88         0
89     }
90 }
```

Listing 2.12: appchain-anchor/src/assets/near_fungible_tokens.rs

Suggestion I Set an expiration time for token prices. If the token price is checked to be expired, function calls associated with the token prices should be blocked.

Feedback from the Project As it is not necessary to maintain the price frequently, this optimization can be hold for now. Our supporting team will track this manually.

2.2.3 Callback Functions are Missing

Status Confirmed

Introduced by `version 1`

Description Callback functions are not implemented for these cross-contract invocations listed below.

```
526    // Code snippet of function withdraw_stake
527            if balance_to_withdraw > 0 {
528                ext_ft_core::ext(self.oct_token.get().unwrap().contract_account)
529                    .with_attached_deposit(1)
530                    .with_static_gas(Gas::ONE_TERA.mul(T_GAS_FOR_FT_TRANSFER))
531                    .with_unused_gas_weight(0)
532                    .ft_transfer(account_id, balance_to_withdraw.into(), None);
533            }
534        };
535    }
```

Listing 2.13: appchain-anchor/src/user_actions/staking.rs

```
564    // Code snippet of function withdraw_validator_rewards
565    if reward_to_withdraw > 0 {
566        ext_ft_core::ext(
567            self.wrapped_appchain_token
568                .get()
569                .unwrap()
570                .contract_account
571                .unwrap(),
572        )
573        .with_attached_deposit(1)
574        .with_static_gas(Gas::ONE_TERA.mul(T_GAS_FOR_FT_TRANSFER))
575        .with_unused_gas_weight(0)
576        .ft_transfer(validator_id, reward_to_withdraw.into(), None);
577    }
```

Listing 2.14: appchain-anchor/src/user_actions/staking.rs

```
611    // Code snippet of function withdraw_delegator_rewards
612    if reward_to_withdraw > 0 {
613        ext_ft_core::ext(
614            self.wrapped_appchain_token
615                .get()
616                .unwrap()
617                .contract_account
618                .unwrap(),
619        )
620        .with_attached_deposit(1)
621        .with_static_gas(Gas::ONE_TERA.mul(T_GAS_FOR_FT_TRANSFER))
622        .with_unused_gas_weight(0)
623        .ft_transfer(delegator_id, reward_to_withdraw.into(), None);
624    }
```

Listing 2.15: appchain-anchor/src/user_actions/staking.rs

Impact User's funds may be lost if the cross-contract invocations fail without restoring the state.

Suggestion I Implement corresponding callback functions for each cross-contract invocation.

Feedback from the Project As our design, only the validators and delegators can call this function to withdraw their unbonded stake, the failures will be limited to very extreme cases (such as they delete the

account they used to deposit OCT token). And if it happens, we can handle it manually based on our supporting process. Our frontend calls `storage_deposit` of `wrapped appchain token contract` to ensure it will not failed. And if it happens, we can handle it manually based on our supporting process.

2.2.4 Tokens may be Locked When Bridging is Closed

Status Fixed in `version 2`

Introduced by `version 1`

Description The `near_fungible_token.bridging_state` is not checked in function `internal_process_near_fungible_token_deposit`. Given this, users can still deposit near-fungible tokens into this contract when the token's `Bridging_state` is closed and the appchain may not respond to the token bridging requests.

```
205    pub fn internal_process_near_fungible_token_deposit(
206        &mut self,
207        predecessor_account_id: AccountId,
208        sender_id: AccountId,
209        amount: U128,
210        deposit_message: FTDepositMessage,
211    ) -> PromiseOrValue<U128> {
212        let mut near_fungible_tokens = self.near_fungible_tokens.get().unwrap();
213        if let Some(mut near_fungible_token) =
214            near_fungible_tokens.get_by_contract_account(&predecessor_account_id)
215        {
216            match deposit_message {
217                FTDepositMessage::BridgeToAppchain {
218                    receiver_id_in_appchain,
219                } => {
```

Listing 2.16: appchain-anchor/src/assets/near_fungible_tokens.rs

The same problem exists in function `internal_process_nft_transfer` for bridging the NFT token to the appchain.

```
370pub fn internal_process_nft_transfer(
371    &mut self,
372    predecessor_account_id: AccountId,
373    sender_id: AccountId,
374    nft_owner_id: AccountId,
375    token_id: TokenId,
376    transfer_message: NFTTransferMessage,
377) -> PromiseOrValue<bool> {
378    let mut wrapped_appchain_nfts = self.wrapped_appchain_nfts.get().unwrap();
379    if let Some(mut wrapped_appchain_nft) =
380        wrapped_appchain_nfts.get_by_contract_account(&predecessor_account_id)
381    {
382        match transfer_message {
383            NFTTransferMessage::BridgeToAppchain {
384                receiver_id_in_appchain,
385            } => {
```

Listing 2.17: appchain-anchor/src/assets/wrapped_appchain_nfts.rs

Impact User's transfered tokens may be locked when its `Bridging_state` is closed.

Suggestion I Bridge tokens to the appchain only when the `BridgingState` is active.

2.2.5 Incorrect Wrapped Appchain Token Management

Status Confirmed

Introduced by `version 1`

Description I The `total_supply` of the `wrapped_appchain_token` is not updated when minting or burning tokens.

```
223 fn resolve_wrapped_appchain_token_burning(
224     &mut self,
225     sender_id_in_near: AccountId,
226     receiver_id_in_appchain: String,
227     amount: U128,
228 ) {
229     assert_self();
230     let mut wrapped_appchain_token = self.wrapped_appchain_token.get().unwrap();
231     match env::promise_result(0) {
232         PromiseResult::NotReady => unreachable!(),
233         PromiseResult::Successful(_) => {
234             wrapped_appchain_token.changed_balance = I128::from(
235                 wrapped_appchain_token.changed_balance.0 - i128::try_from(amount.0).unwrap(),
236             );
237             self.wrapped_appchain_token.set(&wrapped_appchain_token);
238             let appchain_notification_history = self.internal_append_appchain_notification(
239                 AppchainNotification::WrappedAppchainTokenBurnt {
240                     sender_id_in_near: sender_id_in_near.clone(),
241                     receiver_id_in_appchain: receiver_id_in_appchain.clone(),
242                     amount: U128::from(amount),
243                 },
244             );
245             log!(
246                 "Wrapped appchain token burnt in contract '{}' by '{}' for '{}' of appchain. Amount
                     : '{}', Crosschain notification index: '{}'.",
247                 &wrapped_appchain_token.contract_account.unwrap(),
248                 &sender_id_in_near,
249                 &receiver_id_in_appchain,
250                 &amount.0,
251                 &appchain_notification_history.index.0
252             );
253         }
```

Listing 2.18: appchain-anchor/src/assets/wrapped_appchain_token.rs

Impact I The total supply of wrapped appchain token maintained in the AppchainAnchor may be inconsistent with that in the appchain.

Suggestion I Updated the `wrapped_appchain_token.total_supply` when minting or burning tokens.

Feedback from the Project I The `total_supply` of wrapped appchain token is controlled by the appchain rather than anchor contract. So it is maintained manually for now. Now the `premined_balance` and

`changed_balance` are used to record the total amount of wrapped appchain token on NEAR side. Before some kinds of new appchain messages are added to sync the real amount from appchain, this value can not be calculated automatically in anchor contract.

Description II When minting the `era_reward` in function `internal_start_distributing _reward_of_era`, the `wrapped_appchain_token.total_supply` is kept still while the `wrapped_appchain_token.changed_balance` is increased by the number of `era_reward`.

```
72  // Code snippet of function internal_start_distributing_reward_of_era
73  // Mint ‘total_reward‘ in the contract of wrapped appchain token.
74  let appchain_settings = self.appchain_settings.get().unwrap();
75  let mut result = self.internal_mint_wrapped_appchain_token(
76      None,
77      env::current_account_id(),
78      appchain_settings.era_reward,
79      appchain_message_nonce,
80      processing_context,
81  );
```

Listing 2.19: appchain-anchor/src/permissionless_actions/distributing_rewards.rs

Impact II This will result in the same amount of token (`era_reward`) locked in the appchain which cannot be bridged from, mainly due to the check listed below in lines 388-391:

```
372  // Code snippet of function internal_stage_appchain_messages
373  AppchainEvent::NativeTokenLocked {
374      owner_id_in_appchain,
375      receiver_id_in_near,
376      amount,
377  } => {
378      if self.asset_transfer_is_paused {
379          let message = format!("Asset transfer is now paused.");
380          let result = AppchainMessageProcessingResult::Error {
381              nonce: appchain_message.nonce,
382              message: message.clone(),
383          };
384          self.record_appchain_message_processing_result(&result);
385          return MultiTxsOperationProcessingResult::Error(message);
386      }
387      let wrapped_appchain_token = self.wrapped_appchain_token.get().unwrap();
388      if i128::try_from(wrapped_appchain_token.premined_balance.0).unwrap()
389          + wrapped_appchain_token.changed_balance.0
390          + i128::try_from(amount.0).unwrap()
391          > i128::try_from(wrapped_appchain_token.total_supply.0).unwrap()
392      {
393          let message = format!("Too much wrapped appchain token to mint.");
394          let result = AppchainMessageProcessingResult::Error {
395              nonce: appchain_message.nonce,
396              message: message.clone(),
397          };
398          self.record_appchain_message_processing_result(&result);
399          return MultiTxsOperationProcessingResult::Error(message);
400      }
```

Listing 2.20: appchain-anchor/src/permissionless_actions/mod.rs

Suggestion II Update the `wrapped_appchain_token.total_supply` when minting `era_reward`.

Feedback from the Project II This is by design. No need to change.

Description III The AppchainAnchor may not be able to mint tokens when applying the `AppchainEvent` of type `NativeTokenLocked` since the `premined_balance` and the `total_supply` are set with the same value in function `sync_basedata_of_wrapped_appchain_token` (lines 84-85).

```
67  fn sync_basedata_of_wrapped_appchain_token(
68      &mut self,
69      metadata: FungibleTokenMetadata,
70      premined_beneficiary: AccountId,
71      premined_balance: U128,
72  ) {
73      self.assert_contract_account_of_wrapped_appchain_token_is_set();
74      let mut wrapped_appchain_token = self.wrapped_appchain_token.get().unwrap();
75      let contract_account = wrapped_appchain_token.contract_account.clone().unwrap();
76      assert_eq!(
77          env::predecessor_account_id(),
78          contract_account,
79          "Only '{}' can call this function.",
80          contract_account
81      );
82      wrapped_appchain_token.metadata = metadata;
83      wrapped_appchain_token.premined_beneficiary = Some(premined_beneficiary);
84      wrapped_appchain_token.premined_balance = premined_balance;
85      wrapped_appchain_token.total_supply = premined_balance;
86      self.wrapped_appchain_token.set(&wrapped_appchain_token);
87  }
```

Listing 2.21: appchain-anchor/src/assets/wrapped_appchain_token.rs

Impact III The check listed below (lines 388-391) may not meet in this case, which may lead to a potential DoS problem.

```
372  // Code snippet of function internal_stage_appchain_messages
373  AppchainEvent::NativeTokenLocked {
374      owner_id_in_appchain,
375      receiver_id_in_near,
376      amount,
377  } => {
378      if self.asset_transfer_is_paused {
379          let message = format!("Asset transfer is now paused.");
380          let result = AppchainMessageProcessingResult::Error {
381              nonce: appchain_message.nonce,
382              message: message.clone(),
383          };
384          self.record_appchain_message_processing_result(&result);
385          return MultiTxsOperationProcessingResult::Error(message);
386      }
387      let wrapped_appchain_token = self.wrapped_appchain_token.get().unwrap();
```

```
388         if i128::try_from(wrapped_appchain_token.premined_balance.0).unwrap()
389             + wrapped_appchain_token.changed_balance.0
390             + i128::try_from(amount.0).unwrap()
391             > i128::try_from(wrapped_appchain_token.total_supply.0).unwrap()
392         {
393             let message = format!("Too much wrapped appchain token to mint.");
394             let result = AppchainMessageProcessingResult::Error {
395                 nonce: appchain_message.nonce,
396                 message: message.clone(),
397             };
398             self.record_appchain_message_processing_result(&result);
399             return MultiTxsOperationProcessingResult::Error(message);
400         }
```

Listing 2.22: appchain-anchor/src/permissionless_actions/mod.rs

Suggestion III Update the `wrapped_appchain_token.total_supply` with a reasonable value after activating the appchain.

Feedback from the Project III This is also intentional. The admin should set the `total_supply` manually after activating the appchain. The actual value is dependent on the settings in appchain.

2.3 Additional Recommendation

2.3.1 Redundant Code

Status Fixed in `version 2`

Introduced by `version 1`

Description The existence of the`token_price_maintainer_account` is repeatedly checked in both line 322 and line 326 in function `assert_token_price_maintainer`.

```
319     fn assert_token_price_maintainer(&self) {
320         let anchor_settings = self.anchor_settings.get().unwrap();
321         assert!(
322             anchor_settings.token_price_maintainer_account.is_some(),
323             "Token price maintainer account is not set."
324         );
325         let token_price_maintainer_account =
326             anchor_settings.token_price_maintainer_account.unwrap();
327         assert_eq!(
328             env::predecessor_account_id(),
329             token_price_maintainer_account,
330             "Only '{}' can call this function.",
331             token_price_maintainer_account
332         );
333     }
```

Listing 2.23: appchain-anchor/src/lib.rs

The same problem exists in another function `assert_relayer`.

```
335     fn assert_relayer(&self) {
336         let anchor_settings = self.anchor_settings.get().unwrap();
```

```
337         assert!(
338             anchor_settings.relayer_account.is_some(),
339             "Relayer account is not set."
340         );
341         let relayer_account = anchor_settings.relayer_account.unwrap();
342         assert_eq!(
343             env::predecessor_account_id(),
344             relayer_account,
345             "Only '{}' can call this function.",
346             relayer_account
347         );
348     }
```

Listing 2.24: appchain-anchor/src/lib.rs

Suggestion I It is recommended to implement the `expect(msg:&str)` pattern with custom messages for the panics.

2.3.2 Assertion Missing for Changing NEAR Fungible Token Metadata

Status Fixed in `version 2`

Introduced by `version 1`

Description When changing the near-fungible token metadata, there is no check on the existence of `contract_account` in function `change_near_fungible_token_metadata`. Without the check, two registered near-fungible tokens with different symbols may refer to the same token `contract_account`.

```
136     fn change_near_fungible_token_metadata(
137         &mut self,
138         symbol: String,
139         name: String,
140         decimals: u8,
141         contract_account: AccountId,
142     ) {
143         self.assert_owner();
144         let mut near_fungible_tokens = self.near_fungible_tokens.get().unwrap();
145         assert!(
146             near_fungible_tokens.contains(&symbol),
147             "Token '{}' is not registered.",
148             &symbol
149         );
150         let mut near_fungible_token = near_fungible_tokens.get(&symbol).unwrap();
151         near_fungible_token.metadata.name = name;
152         near_fungible_token.metadata.decimals = decimals;
153         near_fungible_token.contract_account = contract_account;
154         near_fungible_tokens.insert(&near_fungible_token);
155     }
```

Listing 2.25: appchain-anchor/src/assets/near_fungible_tokens.rs

Suggestion I Add the check described above in function `change_near_fungible_token_metadata`.

2.3.3 Potential Precision Loss

Status Confirmed

Introduced by version 1

Description In function `distribute_reward_in_validator_set`, the division is performed before multiplication when calculating the corresponding value, which may result in precision loss.

```
254    if delegator_index >= validator_set.get_delegator_count_of(&validator.validator_id) {
255        let validator_reward = validator_commission_reward
256            + (total_reward_of_validator - validator_commission_reward)
257                * (validator.deposit_amount / OCT_DECIMALS_VALUE)
258                / (validator.total_stake / OCT_DECIMALS_VALUE);
```

Listing 2.26: appchain-anchor/src/permissionless_actions/distributing_rewards.rs

Suggestion I Modify this calculation to perform multiplication before division and map the variables in the calculation into integer type `U256` for avoiding overflow.

Feedback from the Project As current implementation is already active on our mainnet, I think the precision loss is acceptable to users. I prefer to keep current implementation.

2.3.4 Potential Unreasonable Protocol Settings

Status Fixed in version 2

Introduced by version 1

Description The `protocol_settings.minimum_validator_deposit` is recommended to be greater than the `protocol_settings.minimum_validator_deposit_changing_amount`.

```
55    fn change_minimum_validator_deposit(&mut self, value: U128) {
56        self.assert_owner();
57        let mut protocol_settings = self.protocol_settings.get().unwrap();
58        assert!(
59            value.0 != protocol_settings.minimum_validator_deposit.0,
60            "The value is not changed."
61        );
62        protocol_settings.minimum_validator_deposit = value;
63        self.protocol_settings.set(&protocol_settings);
64    }
65    //
66    fn change_minimum_validator_deposit_changing_amount(&mut self, value: U128) {
67        self.assert_owner();
68        let mut protocol_settings = self.protocol_settings.get().unwrap();
69        assert!(
70            value.0
71                != protocol_settings
72                    .minimum_validator_deposit_changing_amount
73                    .0,
74            "The value is not changed."
75        );
76        protocol_settings.minimum_validator_deposit_changing_amount = value;
77        self.protocol_settings.set(&protocol_settings);
78    }
```

Listing 2.27: appchain-anchor/src/user_actions/settings_manager.rs

The same problem exists in the protocol settings associated with the delegator's deposit.

```
91  fn change_minimum_delegator_deposit(&mut self, value: U128) {
92      self.assert_owner();
93      let mut protocol_settings = self.protocol_settings.get().unwrap();
94      assert!(
95          value.0 != protocol_settings.minimum_delegator_deposit.0,
96          "The value is not changed."
97      );
98      protocol_settings.minimum_delegator_deposit = value;
99      self.protocol_settings.set(&protocol_settings);
100 }
101 //
102 fn change_minimum_delegator_deposit_changing_amount(&mut self, value: U128) {
103     self.assert_owner();
104     let mut protocol_settings = self.protocol_settings.get().unwrap();
105     assert!(
106         value.0
107             != protocol_settings
108                 .minimum_delegator_deposit_changing_amount
109                 .0,
110         "The value is not changed."
111     );
112     protocol_settings.minimum_delegator_deposit_changing_amount = value;
113     self.protocol_settings.set(&protocol_settings);
114 }
```

Listing 2.28: appchain-anchor/src/user_actions/settings_manager.rs

Suggestion I Ensure that the `changing_amount` is greater than the `minimum_value` in these functions listed above.

2.3.5 Potential Inconsistencies between Code and Storage

Status Fixed in `version 2`

Introduced by `version 1`

Description According to the current implementation of contract upgradation, the deployment of the new version of the contract code and the migration of the storage are separated into different transactions. There is a potential DoS problem caused by the inconsistencies between the code and storage if the owner does not migrate the storage in time.

```
70  #[init(ignore_state)]
71  pub fn migrate_state() -> Self {
72      // Deserialize the state using the old contract structure.
73      let old_contract: OldAppchainAnchor = env::state_read().expect("Old state doesn't exist");
74      // Verify that the migration can only be done by the owner.
75      // This is not necessary, if the upgrade is done internally.
76      assert_eq!(
77          &env::predecessor_account_id(),
```

```
78              &old_contract.owner,
79              "Can only be called by the owner"
80          );
81          //
82          // Create the new contract using the data from the old contract.
83          let new_contract = AppchainAnchor {
84              appchain_id: old_contract.appchain_id,
85              appchain_registry: old_contract.appchain_registry,
86              owner: old_contract.owner,
87              owner_pk: env::signer_account_pk(),
88              oct_token: old_contract.oct_token,
89              wrapped_appchain_token: old_contract.wrapped_appchain_token,
90              near_fungible_tokens: old_contract.near_fungible_tokens,
91              validator_set_histories: old_contract.validator_set_histories,
92              next_validator_set: old_contract.next_validator_set,
93              unwithdrawn_validator_rewards: old_contract.unwithdrawn_validator_rewards,
94              unwithdrawn_delegator_rewards: old_contract.unwithdrawn_delegator_rewards,
95              unbonded_stakes: old_contract.unbonded_stakes,
96              validator_profiles: old_contract.validator_profiles,
97              appchain_settings: old_contract.appchain_settings,
98              anchor_settings: old_contract.anchor_settings,
99              protocol_settings: old_contract.protocol_settings,
100             appchain_state: old_contract.appchain_state,
101             staking_histories: old_contract.staking_histories,
102             anchor_event_histories: old_contract.anchor_event_histories,
103             appchain_notification_histories: old_contract.appchain_notification_histories,
104             permissionless_actions_status: old_contract.permissionless_actions_status,
105             beefy_light_client_state: old_contract.beefy_light_client_state,
106             reward_distribution_records: old_contract.reward_distribution_records,
107             asset_transfer_is_paused: old_contract.asset_transfer_is_paused,
108             user_staking_histories: old_contract.user_staking_histories,
109             rewards_withdrawal_is_paused: old_contract.rewards_withdrawal_is_paused,
110             appchain_messages: old_contract.appchain_messages,
111             appchain_challenges: old_contract.appchain_challenges,
112             wrapped_appchain_nfts: LazyOption::new(
113                 StorageKey::WrappedAppchainNFTs.into_bytes(),
114                 Some(&WrappedAppchainNFTs::new()),
115             ),
116         };
117         //
118         //
119         new_contract
120     }
```

Listing 2.29: appchain-anchor/src/storage_migration.rs

Suggestion I It is recommended to arrange the deployment of the new version of the contract code and the migration of the storage in one transaction (function).

2.3.6 Missing Check on the Range of the Validator Count

Status Fixed in `version 2`

Introduced by version 1

Description It is recommended to check that the `maximum_validator_count` is greater than the `mininum_validator_count` in both function `change_minimum_validator_count` and `change_maximum_validator_count`.

```
148    //
149    fn change_minimum_validator_count(&mut self, value: U64) {
150        self.assert_owner();
151        let mut protocol_settings = self.protocol_settings.get().unwrap();
152        assert!(
153            value.0 != protocol_settings.minimum_validator_count.0,
154            "The value is not changed."
155        );
156        protocol_settings.minimum_validator_count = value;
157        self.protocol_settings.set(&protocol_settings);
158    }
159    //
160    fn change_maximum_validator_count(&mut self, value: U64) {
161        self.assert_owner();
162        let mut protocol_settings = self.protocol_settings.get().unwrap();
163        assert!(
164            value.0 != protocol_settings.maximum_validator_count.0,
165            "The value is not changed."
166        );
167        protocol_settings.maximum_validator_count = value;
168        self.protocol_settings.set(&protocol_settings);
169    }
```

Listing 2.30: appchain-anchor/src/user_actions/settings_manager.rs

Suggestion I Implementation corresponding assertions in function `change_minimum_validator_count` and `change_maximum_validator_count`.

2.3.7 Potential Centralization Problem

Status Confirmed

Introduced by version 1

Description This project has potential centralization problems. The project owner needs to ensure the security of the private key of the `AppchainAnchor.owner` 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 a multi-signature or a public DAO.

Feedback from the Project Yes, we know the risk, and we take it for now. We have planed to replace the owner by a DAO contract by this year.

2.3.8 Potential Elastic Supply Token Problem

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.

Feedback from the Project Noted. The whitelist is maintained by our team, we'll be care of this case.

2.4 Additional Note

2.4.1 Errors from the Appchain are Ignored

Status Confirmed

Introduced by `version 1`

Description There is no error handlers implemented for the `AppchainNotifications` if they are not properly handled by the external appchain.

Feedback from the Project The `AppchainNotification` is used by the appchain to continue processing the necessary logic. In current design, there is no need to sync the processing result back to anchor contract.