Octopus Network Anchor

NEAR Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **March 8th, 2022 - April 24th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	08/03/2022	Mustafa Hasan
0.2	Document Edit	04/18/2022	Mustafa Hasan
0.3	Document Edit	04/29/2022	Timur Guvenkaya
0.4	Draft Review	04/29/2022	Gabi Urrutia
1.0	Remediation Plan	05/05/2022	Timur Guvenkaya
1.1	Remediation Plan Review	05/11/2022	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Timur Guvenkaya	Halborn	Timur.Guvenkaya@halborn.com
Mustafa Hasan	Halborn	Mustafa.Hasan@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Octopus Network engaged Halborn to conduct a security assessment on their NEAR smart contracts beginning on March 8th, 2022 and ending April 24th, 2022. Octopus Network is a multichain interoperable cryptonetwork for launching and running Web3.0 Substrate-based application-specific blockchains, aka appchains.

Though this security audit's outcome is satisfactory, only the most essential aspects were tested and verified to achieve objectives and deliverables set in the scope due to time and resource constraints. It is essential to note the use of the best practices for secure development.

1.2 AUDIT SUMMARY

The team at Halborn was provided six weeks for the engagement and assigned one full-time security engineer to audit the security of the assets in scope. The engineer is a blockchain and smart contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to achieve the following:

- Identify potential security issues within the NEAR smart contracts.

In summary, Halborn identified few security risks that were mostly addressed by the Octopus Network team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual view of the code and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the smart contract audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of smart contracts and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture, purpose, and use of the platform.
- Manual code read and walkthrough.
- Manual Assessment of use and safety for the critical Rust variables and functions in scope to identify any arithmetic related vulnerability classes.
- Fuzz testing. (cargo fuzz)
- Checking the unsafe code usage. (cargo-geiger)
- Scanning of Rust files for vulnerabilities.(cargo audit)
- Deployment to devnet through near-cli

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

Appchain Anchhor

- Appchain Anchor for-halborn-audit branch

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	6	6	0

LIKELIHOOD

IMPACT

(HAL-01) (HAL-02) (HAL-03)	🟨	🟥	🟥	🟪
🟩	(HAL-05) (HAL-06)	🟨	🟥	🟥
🟩	🟩	(HAL-04)	🟨	🟥
⬜	(HAL-07) (HAL-08) (HAL-09) (HAL-10) (HAL-11) (HAL-12)	🟩	🟨	🟨
⬜	⬜	🟩	🟩	🟨

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL01 - TOKEN PRICE MAINTAINER AS WELL AS RELAYER CAN BE SET TO THE OWNERS ACCOUNTID	Medium	SOLVED - 03/14/2022
HAL02 - OWNER ACCOUNTID CAN BE SET TO AN INVALID VALUE	Medium	SOLVED - 03/14/2022
HAL03 - LACK OF VALIDATION ALLOWS SETTING PERCENTAGES HIGHER THAN A HUNDRED	Medium	PARTIALLY SOLVED
HAL04 - CASE SENSITIVE CHECK ALLOWS ADDING THE SAME NEAR FUNGIBLE TOKEN MORE THAN ONCE	Medium	SOLVED - 03/14/2022
HAL05 - MISSING ZERO CHECKS ON AMOUNTS AND PRICES	Medium	NOT APPLICABLE
HAL06 - LACK OF UPPER LIMIT CHECKS ALLOWS BLOCKING WITHDRAWALS	Medium	RISK ACCEPTED
HAL07 - LACK OF UPPER BOUND CHECKS ON DECIMALS LEADS TO PANICS	Medium	RISK ACCEPTED
HAL08 - MINIMUM VALIDATOR COUNT CAN BE SET TO 0 OR 1	Low	RISK ACCEPTED
HAL09 - DEFAULT BRIDGING STATE FOR NEW NEAR FUNGIBLE TOKENS IS ‘ACTIVE‘ WHILE IT SHOULD BE ‘CLOSED‘	Low	SOLVED - 03/14/2022
HAL10 - LACK OF VALIDATOR ACCOUNTID VALIDATION ALLOWS USING INVALID ACCOUNTIDS	Low	NOT APPLICABLE
HAL11 - NO URL VALIDATION ON SETTING RPC AND SubQL ENDPOINTS	Low	RISK ACCEPTED
HAL12 - LACK OF UPPER LIMIT CHECKS MAY CAUSE RESOURCE EXHAUSTION	Low	RISK ACCEPTED

FINDINGS & TECH DETAILS

3.1 (HAL-01) TOKEN PRICE MAINTAINER AS WELL AS RELAYER CAN BE SET TO THE OWNERS ACCOUNTID - MEDIUM

Description:

The set_token_price_maintainer_account() and set_relayer_account() methods implemented for the AppchainAnchor struct and can be found in appchain-anchor/src/user_actions/settings_manager.rs do not validate the AccountId passed to them to ensure that no matches that of the owner, allowing a malicious owner to bypass the privilege separation point in this case.

Code Location:

Listing 1: appchain-anchor/src/user_actions/settings_manager.rs

```rust
fn set_token_price_maintainer_account(&mut self, account_id: AccountId) {
    self.assert_owner();
    let mut anchor_settings = self.anchor_settings.get().unwrap();
    anchor_settings.token_price_maintainer_account = account_id;
    self.anchor_settings.set(&anchor_settings);
}
```

Listing 2: appchain-anchor/src/user_actions/settings_manager.rs

```rust
fn set_relayer_account(&mut self, account_id: AccountId) {
    self.assert_owner();
    let mut anchor_settings = self.anchor_settings.get().unwrap();
    anchor_settings.relayer_account = account_id;
    self.anchor_settings.set(&anchor_settings);
}
```

Proof of Concept::

The following test case was created as a PoC:

Listing 3

```rust
fn test_set_owner_as_maintainer(){
    let total_supply = common::to_oct_amount(TOTAL_SUPPLY);
    let (root, _, _registry, anchor, _) = common::init(total_supply, false);
    let result = settings_actions::set_token_price_maintainer_account(&root, &anchor, &root);
    result.assert_success();
    let result2 = view!(anchor.get_owner());
    let owner = result2.unwrap_json::<String>();
    let anchor_settings = anchor_viewer::get_anchor_settings(&anchor);
    let maintainer = anchor_settings.token_price_maintainer_account;
    assert!(owner == maintainer);
}
```

Risk Level:

Likelihood - 1
Impact - 5

Recommendation:

The two affected functions must validate that the AccountId passed to them does not match that of the owner and panic otherwise.

Remediation Plan:

SOLVED: The Octopus Network team solved the issue in commit ef2219a37c5be402cec720d9db03501981c2ca80

3.2 (HAL-02) OWNER ACCOUNTID CAN BE SET TO AN INVALID VALUE - MEDIUM

Description:

The set_owner() method implemented in the AppchainAnchor Struct, which can be found in “appchain-anchor/src/lib.rs”, does not validate that the AccountId value passed to it actually contains a valid AccountId following the NEAR’s account ID rules. As a result, an owner who wishes to update pass ownership to another user can erroneously call the function with a string pointing to an invalid NEAR account ID, resulting in complete and irreversible loss of control over the contract from that point forward.

Code Location:

Listing 4: appchain-anchor/src/lib.rs

```rust
fn set_owner(&mut self, owner: AccountId) {
    self.assert_owner();
    self.owner = owner;
}
```

Proof of Concept::

The following is a test case developed as a PoC, notice that the test prints Owner is: th!1!nv@|!d when finished:

Listing 5

```rust
fn test_set_invalid_owner(){
    let total_supply = common::to_oct_amount(TOTAL_SUPPLY);
    let (root, _, _registry, anchor, _) = common::init(total_supply, false);
    anchor.contract.set_owner("test".to_string());
    let result1 = call!(root, anchor.set_owner("th!$_1$_!nv@|!d".to_string()));
    result1.assert_success();
    let result = view!(anchor.get_owner());
    println!("New owner is: {}", result.unwrap_json::<String>());
}
```

Risk Level:

Likelihood - 1
Impact - 5

Recommendation:

The function must validate that the passed argument is in the form of a valid AccountId before setting the owner.

Remediation Plan:

SOLVED: The Octopus Network team solved the issue in commit ef2219a37c5be402cec720d9db03501981c2ca80

3.3 (HAL-03) LACK OF VALIDATION ALLOWS SETTING PERCENTAGES HIGHER THAN A HUNDRED - MEDIUM

Description:

The change_maximum_validator_stake_percent() method in "appchain-anchor/src/user_actions/settings_manager.rs" checks that the percentage value passed to it is less than a 100 and reverts otherwise. However, all the remaining functions allowing the owner to change other percentage values do not perform such checks, allowing percentages to exceed 100%, which would probably cause the contract to crash and panic while rewards are being distributed.

Code Location:

Listing 6: appchain-anchor/src/user_actions/settings_manager.rs

```rust
fn change_maximum_market_value_percent_of_near_fungible_tokens(&mut self, value: u16) {
    self.assert_owner();
    let mut protocol_settings = self.protocol_settings.get().unwrap();
    assert!(
        value != protocol_settings.maximum_market_value_percent_of_near_fungible_tokens,
        "The value is not changed."
    );
    protocol_settings.maximum_market_value_percent_of_near_fungible_tokens = value;
    self.protocol_settings.set(&protocol_settings);
}
```

Listing 7: appchain-anchor/src/user_actions/settings_manager.rs

```rust
fn change_maximum_market_value_percent_of_wrapped_appchain_token(&mut self, value: u16) {
    self.assert_owner();
    let mut protocol_settings = self.protocol_settings.get().unwrap();
    assert!(
        value != protocol_settings.maximum_market_value_percent_of_wrapped_appchain_token,
        "The value is not changed."
    );
    protocol_settings.maximum_market_value_percent_of_wrapped_appchain_token = value;
    self.protocol_settings.set(&protocol_settings);
}
```

Listing 8: appchain-anchor/src/user_actions/settings_manager.rs

```
fn change_validator_commission_percent(&mut self, value: u16) {
    self.assert_owner();
    let mut protocol_settings = self.protocol_settings.get().unwrap();
    assert!(
        value != protocol_settings.maximum_market_value_percent_of_near_fungible_tokens,
        "The value is not changed."
    );
    protocol_settings.maximum_market_value_percent_of_near_fungible_tokens = value;
    self.protocol_settings.set(&protocol_settings);
}
```

Risk Level:

Likelihood - 1
Impact - 5

Recommendation:

Percentage values should always be checked before assigning them to avoid exceeding allowable levels.

Remediation Plan:

PARTIALLY SOLVED: The Octopus Network team partially solved the issue in commit ef2219a37c5be402cec720d9db03501981c2ca80

Then some checks were removed in commit eaa2a5109bca0522f6a285f53ebe1e366475bbc6

3.4 (HAL-04) CASE SENSITIVE CHECK ALLOWS ADDING THE SAME NEAR FUNGIBLE TOKEN MORE THAN ONCE - MEDIUM

Description:

The register_near_fungible_token() function in "appchain-anchor/src/assets/near_fungible_tokens.rs" only checks if the symbol of the token passed to it already exists, however the check is case-sensitive, so it can be bypassed. This allows the owner to register the same token more than once, which can lead to users distributing their funds under different NEAR token contracts instead of one, reducing liquidity and the rewards.

Code Location:

Listing 9: appchain-anchor/src/assets/near_fungible_tokens.rs

```rust
fn register_near_fungible_token(
    &mut self,
    symbol: String,
    name: String,
    decimals: u8,
    contract_account: AccountId,
    price: U128,
) {
    self.assert_owner();
    let mut near_fungible_tokens = self.near_fungible_tokens.get().unwrap();
    assert!(
        !near_fungible_tokens.contains(&symbol),
        "Token '{}' is already registered.",
        &symbol
    );
    near_fungible_tokens.insert(&NearFungibleToken {
        metadata: FungibleTokenMetadata {
            spec: "ft-1.0.0".to_string(),
            symbol,
            name,
            decimals,
            icon: None,
            reference: None,
            reference_hash: None,
        },
        contract_account,
        price_in_usd: price,
        locked_balance: U128::from(0),
        bridging_state: BridgingState::Active,
    });
    self.near_fungible_tokens.set(&near_fungible_tokens);
}
```

Proof of Concept::

The following test case reproduces the issue and prints the two tokens that were registered with similar names:

Listing 10

```
#[test]
fn test_same_token_registeration(){
    let total_supply = common::to_oct_amount(TOTAL_SUPPLY);
    let (root, _, _registry, anchor, _) = common::init(total_supply, false);
    let result = call!(root, anchor.register_near_fungible_token(
        "HLB".to_string(), "Halborn".to_string(),
        2, "test1".to_string(), U128::from(1)
    ));
    result.assert_success();
    let result = call!(root, anchor.register_near_fungible_token(
        "hlb".to_string(), "halborn".to_string(),
        2, "test2".to_string(), U128::from(5)
    ));
    result.assert_success();
    let result2 = view!(anchor.get_near_fungible_tokens());
    let tokens = result2.unwrap_json::<Vec<NearFungibleToken>>();
    tokens.iter().for_each(|token|{
        println!("Token name: {}, symbol: {}", token.metadata.name, token.metadata.symbol);
    });
}
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

The function must validate that the passed token name is not the same as any of the existing tokens before adding it.

Remediation Plan:

SOLVED: The Octopus Network team solved the issue in commit ef2219a37c5be402cec720d9db03501981c2ca80

3.5 (HAL-05) MISSING ZERO CHECKS ON AMOUNTS AND PRICES - MEDIUM

Description:

Checks should be implemented on amount and price values to make sure they are not set to invalid values, including setting such fields to zero. The set_price_of_oct_token() method implemented in the AppchainAnchor struct in "appchain-anchor/src/lib.rs" does not employ such checks to validate that the price of the OCT token does not drop to 0.

Code Location:

Listing 11: appchain-anchor/src/lib.rs

```
pub fn set_price_of_oct_token(&mut self, price: U128) {
    let anchor_settings = self.anchor_settings.get().unwrap();
    assert_eq!(
        env::predecessor_account_id(),
        anchor_settings.token_price_maintainer_account,
        "Only '{}' can call this function.",
        anchor_settings.token_price_maintainer_account
    );
    let mut oct_token = self.oct_token.get().unwrap();
    oct_token.price_in_usd = price;
    self.oct_token.set(&oct_token);
}
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendation:

A default minimum amount must be checked before setting the price value passed to the function.

Remediation Plan:

NOT APPLICABLE: The Octopus Network team marked the issue as not applicable, as setting OCT to 0 is needed to remove the cross-chain asset transfer restriction.

3.6 (HAL-06) LACK OF UPPER LIMIT CHECKS ALLOWS BLOCKING WITHDRAWALS - MEDIUM

Description:

The change_unlock_period_of_delegator_deposit() and change_unlock_period_of_validato () functions in "appchain-anchor/src/user_actions/settings_manager.rs" do not check for an upper bound for the values passed to them. These functions allow the owner to set the number of days before validators/delegators can withdraw their rewards.

By not checking for an upper bound, the owner can set the values to big numbers that would correspond to years before validators/delegators can actually withdraw their balances.

Code Location:

Listing 12: appchain-anchor/src/user_actions/settings_manager.rs

```rust
fn change_unlock_period_of_delegator_deposit(&mut self, value: U64) {
    self.assert_owner();
    let mut protocol_settings = self.protocol_settings.get().unwrap();
    assert!(
        value.0 != protocol_settings.unlock_period_of_delegator_deposit.0,
        "The value is not changed."
    );
    protocol_settings.unlock_period_of_delegator_deposit = value;
    self.protocol_settings.set(&protocol_settings);
}
```

Listing 13: appchain-anchor/src/user_actions/settings_manager.rs

```rust
fn change_unlock_period_of_validator_deposit(&mut self, value: U64) {
    self.assert_owner();
    let mut protocol_settings = self.protocol_settings.get().unwrap();
    assert!(
        value.0 != protocol_settings.unlock_period_of_validator_deposit.0,
        "The value is not changed."
    );
    protocol_settings.unlock_period_of_validator_deposit = value;
    self.protocol_settings.set(&protocol_settings);
}
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendation:

Upper limits should be checked before setting the value.

Remediation Plan:

RISK ACCEPTED: The Octopus Network team accepted the risk of this finding.

3.7 (HAL-07) MINIMUM VALIDATOR COUNT CAN BE SET TO 0 OR 1 - LOW

Description:

The change_minimum_validator_count() method in "appchain-anchor/src/user_actions/settings_manager.rs" does not guarantee that the minimum validator count set by the owner is not less than 0 or 1 validators at the same time, that directly impacts the security of the chain. A minimum validators constant should be set and checked in that function to make sure validator counts cannot go below that threshold.

Code Location:

Listing 14: appchain-anchor/src/user_actions/settings_manager.rs

```
fn change_minimum_validator_count(&mut self, value: U64) {
    self.assert_owner();
    let mut protocol_settings = self.protocol_settings.get().unwrap();
    assert!(
        value.0 != protocol_settings.minimum_validator_count.0,
        "The value is not changed."
    );
    protocol_settings.minimum_validator_count = value;
    self.protocol_settings.set(&protocol_settings);
}
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

The minimum validator count must be compared to a fixed lower amount before being set.

Remediation Plan:

RISK ACCEPTED: The Octopus Network team accepted the risk stating "The minimum validator count of the protocol settings will only affect the verification process in the go_booting function, it is just a reference value. Other than that, it does not affect any other functions in the contract or the actual actions/status of the application chain. In some special cases, we can set it to 0 or 1, to change the state of the contract for further operations."

3.8 (HAL-08) LACK OF UPPER BOUND CHECKS ON DECIMALS LEADS TO PANICS - LOW

Description:

The get_market_value_of() function in ”appchain-anchor/src/assets/near_fungible_tokens.rs” and ”appchain-anchor/src/assets/wrapped_appchain_token.rs” uses the decimals value in the respective token as an exponent while raising a power and the value is then stored in a u128 variable. The issue here is that the decimals value is not validated before being used when the token is created, leading to the ability to pass any value in the range of 0-255, which allows the token creator to cause a panic case whenever the get_market_value_of() function is called since the pow() function call would yield a value well beyond the possible range of u128.

Code Location:

Listing 15: appchain-anchor/src/assets/near_fungible_tokens.rs

```
pub fn get_market_value_of(&self, symbol: &String, amount: u128)
↳ -> Balance {
    if let Some(near_fungible_token) = self.tokens.get(&symbol) {
        amount / u128::pow(10, u32::from(near_fungible_token.
↳ metadata.decimals))
            * near_fungible_token.price_in_usd.0
    } else {
        0
    }
}
```

Listing 16: appchain-anchor/src/assets/wrapped_appchain_token.rs

```
pub fn get_market_value_of(&self, amount: u128) -> Balance {
    amount / u128::pow(10, u32::from(self.metadata.decimals)) *
↳ self.price_in_usd.0
}
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Upper bounds should be checked before setting the value.

Remediation Plan:

RISK ACCEPTED: The Octopus Network team accepted the risk of this finding.

3.9 (HAL-09) DEFAULT BRIDGING STATE FOR NEW NEAR FUNGIBLE TOKENS IS ‘ACTIVE‘ WHILE IT SHOULD BE ‘CLOSED‘ - LOW

Description:

The implementation detail document states that newly registered NEAR fungible tokens should have their bridging state set to Closed. However, the register_near_fungible_token() function in ”appchain-anchor/src/assets/near_fungible_tokens.rs” registers the token with the bridging state of Active.

Code Location:

Listing 17: appchain-anchor/src/assets/near_fungible_tokens.rs

```rust
fn register_near_fungible_token(
    &mut self,
    symbol: String,
    name: String,
    decimals: u8,
    contract_account: AccountId,
    price: U128,
) {
    self.assert_owner();
    let mut near_fungible_tokens = self.near_fungible_tokens.get()
 ↳ .unwrap();
    assert!(
        !near_fungible_tokens.contains(&symbol),
        "Token '{}' is already registered.",
        &symbol
    );
    near_fungible_tokens.insert(&NearFungibleToken {
        metadata: FungibleTokenMetadata {
            spec: "ft-1.0.0".to_string(),
            symbol,
            name,
            decimals,
            icon: None,
            reference: None,
            reference_hash: None,
        },
        contract_account,
        price_in_usd: price,
        locked_balance: U128::from(0),
        bridging_state: BridgingState::Active,
    });
    self.near_fungible_tokens.set(&near_fungible_tokens);
}
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

The function should set the bridging_state value to BridgingState::Closed instead of BridgingState::Active while creating the token.

Remediation Plan:

SOLVED: The Octopus Network team solved the issue in commit ef2219a37c5be402cec720d9db03501981c2ca80

3.10 (HAL-10) LACK OF VALIDATOR ACCOUNTID VALIDATION ALLOWS USING INVALID ACCOUNTID - LOW

Description:

The force_change_account_id_in_appchain_of_staking_history() function in "appchain-anchor/src/user_actions/sudo_actions.rs" doesn't validate the account_id_in_appchain value passed to it before using it to register a new validator, allowing the owner to register validators with invalid account IDs.

Code Location:

Listing 18: appchain-anchor/src/user_actions/sudo_actions.rs

```
fn force_change_account_id_in_appchain_of_staking_history(
    &mut self,
    index: U64,
    account_id_in_appchain: String,
) {
    self.assert_owner();
    let mut staking_histories = self.staking_histories.get().unwrap();
    if let Some(mut staking_history) = staking_histories.get(&index.0) {
        match staking_history.staking_fact {
            StakingFact::ValidatorRegistered {
                validator_id,
                validator_id_in_appchain: _,
                amount,
                can_be_delegated_to,
            } => {
                staking_history.staking_fact = StakingFact::ValidatorRegistered {
                    validator_id,
                    validator_id_in_appchain: account_id_in_appchain,
                    amount,
                    can_be_delegated_to,
                };
                staking_histories.insert(&index.0, &staking_history);
                self.staking_histories.set(&staking_histories);
            }
            _ => (),
        }
    }
}
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Checks should exist to validate the passed AccountId value before it is set.

Remediation plan:

NOT APPLICABLE: The Octopus Network team marked the issue as not applicable, stating "Verification of the account_id_in_appchain parameter is not possible. The rule depends on the implementation of the corresponding application chain. And the function is only used in a very rare case and can only be called by the owner."

3.11 (HAL-11) NO URL VALIDATION ON SETTING RPC AND SubQL ENDPOINTS - LOW

Description:

The set_rpc_endpoint() and set_subql_endpoint() functions in "appchain-anchor/src/user_actions/settings_manager.rs" do not validate the values passed before setting them, allowing the owner to set the URLs to any values that may not follow correct URL forms.

Code Location:

Listing 19: appchain-anchor/src/user_actions/settings_manager.rs

```rust
fn set_rpc_endpoint(&mut self, rpc_endpoint: String) {
    self.assert_owner();
    let mut appchain_settings = self.appchain_settings.get().unwrap();
    appchain_settings.rpc_endpoint = rpc_endpoint;
    self.appchain_settings.set(&appchain_settings);
}
//
fn set_subql_endpoint(&mut self, subql_endpoint: String) {
    self.assert_owner();
    let mut appchain_settings = self.appchain_settings.get().unwrap();
    appchain_settings.subql_endpoint = subql_endpoint;
    self.appchain_settings.set(&appchain_settings);
}
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

URLs should be checked to make sure they are valid before they are set.

Remediation Plan:

RISK ACCEPTED: The Octopus Network team accepted the risk of this finding.

3.12 (HAL-12) LACK OF UPPER LIMIT CHECKS MAY CAUSE RESOURCE EXHAUSTION - LOW

Description:

The change_maximum_era_count_of_unwithdrawn_reward() function in "appchain-anchor/src/user_actions/settings_manager.rs" doesn't check for an upper limit to the passed value before setting it, allowing the owner to pass a big enough value that is always higher than the end_era value used subsequently in withdraw_delegator_rewards() and withdraw_validator_rewards() (residing in "appchain-anchor/src/user_actions/staking.rs") and compared with the maximum_era_count_of_unwithdrawn_reward value. The outcome of this behavior would be looping over a bigger space each time each of these functions is called, which would consume more gas.

Code Location:

Listing 20: appchain-anchor/src/user_actions/settings_manager.rs

```
fn change_maximum_era_count_of_unwithdrawn_reward(&mut self, value: U64) {
    self.assert_owner();
    let mut protocol_settings = self.protocol_settings.get().unwrap();
    assert!(
        value.0 != protocol_settings.maximum_era_count_of_unwithdrawn_reward.0,
        "The value is not changed."
    );
    protocol_settings.maximum_era_count_of_unwithdrawn_reward = value;
    self.protocol_settings.set(&protocol_settings);
}
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Upper bounds should be checked before setting the value.

Remediation Plan:

RISK ACCEPTED: The Octopus Network team accepted the risk of this finding.

AUTOMATED TESTING

4.1 AUTOMATED ANALYSIS

Descriptiona:

Halborn used automated security scanners to assist with detection of well-known security issues and vulnerabilities. Among the tools used was cargo audit, a security scanner for vulnerabilities reported to the RustSec Advisory Database. All vulnerabilities published in https://crates.io are stored in a repository named The RustSec Advisory Database. cargo audit is a human-readable version of the advisory database which performs a scanning on Cargo.lock. Security Detections are only in scope. All vulnerabilities shown here were already disclosed in the above report. However, to better assist the developers maintaining this code, the auditors are including the output with the dependencies tree, and this is included in the cargo audit output to better know the dependencies affected by unmaintained and vulnerable crates.

Results:

ID	package	Short Description
RUSTSEC-2020-0159	chrono	Potential segfault in ‘localtime_r‘ invocations
RUSTSEC-2021-0067	cranelift-codegen	Memory access due to code generation flaw in Cranelift module
RUSTSEC-2021-0013	raw-cpuid	Soundness issues in ‘raw-cpuid‘
RUSTSEC-2021-0089	raw-cpuid	Optional ‘Deserialize‘ implementations lacking validation
RUSTSEC-2022-0013	regex	Regexes with large repetitions on empty sub-expressions take a very long time to parse
RUSTSEC-2020-0071	time	Potential segfault in the time crate
RUSTSEC-2021-0110	wasmtime	Multiple Vulnerabilities in Wasmtime

THANK YOU FOR CHOOSING

// HALBORN