Octopus Network

NEAR Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **October 7th, 2021 - December 1st, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	10/7/2021	Timur Guvenkaya
0.2	Document Edit	10/18/2021	Timur Guvenkaya
0.3	Document Edit	10/28/2021	Timur Guvenkaya
0.4	Document Edit	11/5/2021	Timur Guvenkaya
0.5	Document Edit	11/12/2021	Timur Guvenkaya
0.6	Document Edit	11/25/2021	Timur Guvenkaya
0.7	Document Edit	11/29/2021	Timur Guvenkaya
0.8	Draft Review	12/02/2021	Gabi Urrutia
1.0	Remediation Plan	12/07/2021	Timur Guvenkaya
1.1	Remediation Plan Review	12/07/2021	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

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Octopus Network engaged Halborn to conduct a security assessment on their NEAR smart contracts beginning on October 7th, 2021 and ending December 1st, 2021. 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. It's quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that was 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

octopus-appchain-registry

- appchain_anchor_callback.rs
- appchain_basedata.rs
- appchain_owner_actions.rs
- lib.rs
- registry_owner_actions.rs
- registry_settings_actions.rs
- registry_status.rs
- storage_key.rs
- sudo_actions.rs
- types.rs
- upgradable.rs
- voter_actions.rs

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
2	1	4	5	3

LIKELIHOOD

IMPACT

		(HAL-03)		(HAL-01) (HAL-02)
(HAL-09) (HAL-10)		(HAL-04)		
	(HAL-08)	(HAL-05) (HAL-06)	(HAL-07)	
		(HAL-11) (HAL-12)		
(HAL-14) (HAL-15)	(HAL-13)			

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL01 - ANYONE CAN CHANGE OCT TOKEN ACCOUNT	Critical	SOLVED - 10/21/2021
HAL02 - SMART CONTRACT MAIN FUNCTIONALITY DoS	Critical	SOLVED - 12/02/2021
HAL03 - IMPROPER ROLE-BASED ACCESS CONTROL POLICY	High	SOLVED - 12/14/2021
HAL04 - REGISTRY OWNER CAN SET ITSELF AS VOTER OPERATOR	Medium	SOLVED - 11/05/2021
HAL05 - REGISTRY OWNER CAN BE SET AS APPCHAIN OWNER	Medium	PARTIALLY SOLVED
HAL06 - USAGE OF SIGNER ACCOUNT ID INSTEAD OF PREDECESSOR ID IN ACCESS CONTROL	Medium	SOLVED - 12/02/2021
HAL07 - APPCHAIN CAN BE REGISTERED WITHOUT CORE DETAILS	Medium	SOLVED - 11/05/2021
HAL08 - MISSING CARGO OVERFLOW CHECKS	Low	SOLVED - 11/10/2021
HAL09 - LACK OF PAUSABILITY OF SMART CONTRACTS	Low	SOLVED - 12/14/2021
HAL10 - USAGE OF VULNERABLE CRATES	Low	RISK ACCEPTED
HAL11 - MISSING ZERO VALUE CHECK	Low	SOLVED - 12/02/2021
HAL12 - REDUNDANT CODE	Low	SOLVED - 11/05/2021
HAL13 - MISSING REASSIGNMENT CHECKS	Informational	SOLVED - 11/05/2021
HAL14 - CODE REFACTOR OPPORTUNITY	Informational	SOLVED - 12/02/2021
HAL15 - OUTDATED RUST EDITION	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) ANYONE CAN CHANGE OCT TOKEN ACCOUNT - CRITICAL

Description:

It was observed that the change_oct_token is lacking the ownership check, which allows anyone to change the OCT token account.

Code Location:

Listing 1: appchain-registry/src/sudo_actions.rs (Lines 19)

```
17 impl SudoActions for AppchainRegistry {
18     //
19     fn change_oct_token(&mut self, oct_token: AccountId) {
20         self.oct_token = oct_token;
21     }
22
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

It is recommended to add self.assert_owner(); to prevent anyone from changing the OCT token account.

Remediation Plan:

SOLVED: The Octopus Network team solved this issue by removing this function.

3.2 (HAL-02) SMART CONTRACT MAIN FUNCTIONALITY DoS - CRITICAL

Description:

It was observed that the project is vulnerable to DoS of the main functionality. In NEAR, there is a validation that tells whether the account format is valid or not. During conclude_voting_score, the new sub_account is created by appending the appchain_id to the registry account:

Listing 2: appchain-registry/src/registry_owner_actions.rs (Lines 187)

```
180 fn conclude_voting_score(&mut self) {
181         self.assert_owner();
182         assert!(
183             !self.top_appchain_id_in_queue.is_empty(),
184             "There is no appchain on the top of queue yet."
185         );
186         // Set the appchain with the largest voting score to go `
                staging`
187         let sub_account_id = format!(
188             "{}.{}",
189             &self.top_appchain_id_in_queue,
190             env::current_account_id()
191         );
```

Then, at the end, smart contracts creates a new create_account promise action to create new sub account:

Listing 3: appchain-registry/src/registry_owner_actions.rs (Lines 215)

```
214     Promise::new(sub_account_id)
215             .create_account()
216             .transfer(APPCHAIN_ANCHOR_INIT_BALANCE)
217             .add_full_access_key(self.owner_pk.clone());
218     }
```

The issue is that no check ensures that the appchain_id complies with NEAR's validation rules. Therefore, if invalid appchain_id became

top_appchain_id_in_queue and used during the creation of sub_account, the smart contract will inevitably panic during the creation of the account. Since there is no functionality to remove top_appchain_id_in_queue, the smart contract won't conclude votes anymore. The smart contract will get stuck it at that appchain_id.

Code Location:

Listing 4: appchain-registry/src/registry_owner_actions.rs (Lines 215)

```
214    Promise::new(sub_account_id)
215              .create_account()
216              .transfer(APPCHAIN_ANCHOR_INIT_BALANCE)
217              .add_full_access_key(self.owner_pk.clone());
218     }
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

Even though the Octopus Network team reviews appchains and their registration data manually to avoid that, the issue is critical from the smart contract perspective. It is always better to be safe from human error. Therefore, please add account validation during the appchain registration phase to avoid this issue. You can utilize is_top_level_account_id and is_sub_account_of functions within the nearcore.

Remediation Plan:

SOLVED: The Octopus Network team solved the issue by using ValidAccountId helper class.

Fixed Code:

Listing 5: appchain-registry/src/lib.rs (Lines 329,330,331)

```
325         assert!(
326             !appchain_id.trim().is_empty(),
327             "Missing necessary field 'appchain_id'."
328         );
329         assert!(appchain_id.find(".").is_none(), "Invalid '
                appchain_id'.");
330         assert!(
331             ValidAccountId::try_from(format!("{}.{}", appchain_id,
                    env::current_account_id()))
332                 .is_ok(),
333             "Invalid 'appchain_id'."
334         );
```

3.3 (HAL-03) IMPROPER ROLE-BASED ACCESS CONTROL POLICY - HIGH

Description:

It was observed that most of the privileged functionality is controlled by the `owner`. Additional authorization levels are needed to implement the least privilege principle, also known as least-authority, which ensures only authorized processes, users, or programs can access the necessary resources or information. The ownership role is helpful in a simple system, but more complex projects require more roles by using role-based access control.

Code Location:

The owner can access those functions:

- All functions in `sudo_actions.rs`
- All functions in `registry_settings_actions.rs`
- All functions in `registry_owner_actions.rs` except `count_voting_score`
- `set_owner` in `lib.rs`

Risk Level:

Likelihood - 3
Impact - 5

Recommendation:

It is recommended to add additional roles to comply with the least privilege principle and limit the privileges of `owner`.

Remediation Plan:

SOLVED: The Octopus Network team solved the issue by adding role based access control functionality..

3.4 (HAL-04) REGISTRY OWNER CAN SET ITSELF AS VOTER OPERATOR - MEDIUM

Description:

It was observed that the owner could set itself as a voter_operator. This functionality violates the principle of least privilege giving the owner additional privileges.

Code Location:

Listing 6: appchain-registry/src/registry_settings_actions.rs (Lines 57)

```
57    fn change_operator_of_counting_voting_score(&mut self,
          operator_account: AccountId) {
58        self.assert_owner();
59        let mut registry_settings = self.registry_settings.get().
              unwrap();
60        registry_settings.operator_of_counting_voting_score.clear
              ();
61        registry_settings
62            .operator_of_counting_voting_score
63            .push_str(&operator_account);
64        self.registry_settings.set(&registry_settings);
65    }
66
```

Risk Level:

Likelihood - 3
Impact - 4

Recommendation:

It is recommended to add another check to do not allow the owner to set itself as a voter operator.

Remediation Plan:

SOLVED: The Octopus Network team solved the issue by adding relevant check.

Fixed Code:

Listing 7: appchain-registry/src/registry_settings_actions.rs (Lines 61)

```
57     fn change_operator_of_counting_voting_score(&mut self,
           operator_account: AccountId) {
58         self.assert_owner();
59         assert_ne!(
60             operator_account, self.owner,
61             "The account should NOT be the owner."
62         );
63         let mut registry_settings = self.registry_settings.get().
               unwrap();
64         assert_ne!(
65             operator_account, registry_settings.
                   operator_of_counting_voting_score,
66             "The account is not changed."
67         );
68         registry_settings.operator_of_counting_voting_score =
               operator_account;
69         self.registry_settings.set(&registry_settings);
70     }
```

3.5 (HAL-05) REGISTRY OWNER CAN BE SET AS APPCHAIN OWNER - MEDIUM

Description:

It was observed that the owner could be set as an appchain_owner. This functionality violates the principle of least privilege giving the owner additional privileges.

Code Location:

appchain-registry/src/lib.rs: register_appchain
- sender_id should not be equal to the registry owner

appchain-registry/src/appchain_owner_actions.rs: transfer_appchain_ownership

- new_owner should not be equal to the registry owner

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

It is recommended to add an additional check to do not allow the owner to set itself as an voter operator.

Remediation Plan:

PARTIALLY SOLVED: The Octopus Network team partially solved the issue by adding the required check only to **appchain-registry/src/lib.rs**.

Fixed Code:

appchain-registry/src/lib.rs

Listing 8: appchain-registry/src/lib.rs (Lines 308,309,310)

```
291 fn register_appchain(
292         &mut self,
293         sender_id: AccountId,
294         appchain_id: AppchainId,
295         register_deposit: Balance,
296         website_url: String,
297         function_spec_url: String,
298         github_address: String,
299         github_release: String,
300         contact_email: String,
301         premined_wrapped_appchain_token_beneficiary: AccountId,
302         premined_wrapped_appchain_token: U128,
303         ido_amount_of_wrapped_appchain_token: U128,
304         initial_era_reward: U128,
305         fungible_token_metadata: FungibleTokenMetadata,
306         custom_metadata: HashMap<String, String>,
307     ) {
308         assert_ne!(
309             sender_id, self.owner,
310             "The register account should NOT be the contract owner
                .”
311         );
```

3.6 (HAL-06) USAGE OF SIGNER ACCOUNT ID INSTEAD OF PREDECESSOR ID IN ACCESS CONTROL - MEDIUM

Description:

It was observed that the env::signer_account_id() was used in the assert_appchain_owner to assert whether the caller is the appchain_owner.

- env::signer_account_id(): The id of the account that either signed the original transaction or issued the initial cross-contract call.
- env::predecessor_account_id(): The id of the account that was the previous contract in the chain of cross-contract calls. If this is the first contract, it is equal to signer_account_id.

From their definitions above, we can derive that the usage of env::signer_account_id() is risky in access control scenarios. There is a risk that the appchain owner can be phished to sign the cross contract call and hence unknowingly let the malicious contract execute functions in the project's contract under that owner's role.

Code Location:

Listing 9: appchain-registry/src/lib.rs (Lines 178)

```
175 fn assert_appchain_owner(&self, appchain_id: &AppchainId) {
176         let appchain_basedata = self.get_appchain_basedata(
                appchain_id);
177         assert_eq!(
178             env::signer_account_id(),
179             appchain_basedata.owner().clone(),
180             "Function can only be called by appchain owner."
181         );
182     }
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

Consider replacing `env::signer_account_id()` with `env::predecessor_account_id()` to avoid that risk.

Remediation Plan:

SOLVED: The Octopus Network team solved the issue by changing `env::signer_account_id()` to `env::predecessor_account_id()`.

Fixed Code:

Listing 10: appchain-registry/src/lib.rs (Lines 178)

```
175 fn assert_appchain_owner(&self, appchain_id: &AppchainId) {
176         let appchain_basedata = self.get_appchain_basedata(
                appchain_id);
177         assert_eq!(
178             env::predecessor_account_id(),
179             appchain_basedata.owner().clone(),
180             "Function can only be called by appchain owner."
181         );
182     }
```

3.7 (HAL-07) APPCHAIN CAN BE REGISTERED WITHOUT CORE DETAILS - MEDIUM

Description:

It was observed that it is possible to register an appchain without providing any core details such as appchain_id, website_url, and so on. Those details are needed for intended functionality of the application.

Code Location:

Existence of those fields has to be enforced:

appchain-registry/src/lib.rs: register_appchain

- appchain_id
- website_url
- function_spec_url
- github_address
- github_release
- contact_email
- premined_wrapped_appchain_token_beneficiary
- fungible_token_metadata.name
- fungible_token_metadata.symbol

Risk Level:

Likelihood - 4
Impact - 3

Recommendation:

It is recommended to add additional checks to enforces those fields.

Remediation Plan:

SOLVED: The Octopus Network team solved the issue by enforcing required fields.

Fixed Code:

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

```
334 assert!(
335             !appchain_id.trim().is_empty(),
336             "Missing necessary field 'appchain_id'."
337         );
338         assert!(
339             !website_url.trim().is_empty(),
340             "Missing necessary field 'website_url'."
341         );
342         assert!(
343             !function_spec_url.trim().is_empty(),
344             "Missing necessary field 'function_spec_url'."
345         );
346         assert!(
347             !github_address.trim().is_empty(),
348             "Missing necessary field 'github_address'."
349         );
350         assert!(
351             !github_release.trim().is_empty(),
352             "Missing necessary field 'github_release'."
353         );
354         assert!(
355             !contact_email.trim().is_empty(),
356             "Missing necessary field 'contact_email'."
357         );
358         assert!(
359             !premined_wrapped_appchain_token_beneficiary
360                 .trim()
361                 .is_empty(),
362             "Missing necessary field '
                premined_wrapped_appchain_token_beneficiary'."
363         );
364         fungible_token_metadata.assert_valid();
365         assert!(
366             !fungible_token_metadata.name.trim().is_empty(),
367             "Missing necessary field 'fungible token name'."
```

```
368         );
369         assert!(
370             !fungible_token_metadata.symbol.trim().is_empty(),
371             "Missing necessary field 'fungible token symbol'."
372         );
```

3.8 (HAL-08) MISSING CARGO OVERFLOW CHECKS - LOW

Description:

It was observed that there is no overflow-checks=true in Cargo.toml. By default, overflow checks are disabled in optimized release builds. Hence, if there is an overflow in release builds, it will be silenced, leading to unexpected behavior of an application. Even if checked arithmetic is used through checked_*, it is recommended to have that check in Cargo.toml.

Code Location:

- Cargo.toml

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

It is recommended to add overflow-checks=true under your release profile in Cargo.toml.

Remediation Plan:

SOLVED: The Octopus Network team solved the issue by adding overflow-checks=true.

3.9 (HAL-09) LACK OF PAUSABILITY OF SMART CONTRACTS - LOW

Description:

The project lacks ability to pause contracts. It is advised that in case of unexpected events temporarily disable some important functions to prevent further damage.

Risk Level:

Likelihood - 1
Impact - 4

Recommendation:

Consider implementing the pause feature in the smart contracts. Furthermore, it is recommended to add a separate role for being responsible for pausing smart contracts when needed.

Remediation Plan:

SOLVED: The Octopus Network team solved the issue by adding the pausability to smart contracts.

3.10 (HAL-10) USAGE OF VULNERABLE CRATES - LOW

Description:

It was observed that the project uses crates with known vulnerabilities.

Code Location:

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-2020-0071	time	Potential segfault in the time crate
RUSTSEC-2021-0110	wasmtime	Multiple Vulnerabilities in Wasmtime

Risk Level:

Likelihood - 1
Impact - 4

Recommendation:

Even if those vulnerable crates cannot impact the underlying application, it is advised to be aware of them. Also, it is necessary to set up dependency monitoring to always be alerted when a new vulnerability is disclosed in one of the project’s crates.

Remediation Plan:

RISK ACCEPTED: The Octopus Network team acknowledged the issue and is working on fixing it.

3.11 (HAL-11) MISSING ZERO VALUE CHECK - LOW

Description:

There are functions within the project that should have a zero value check.

Code Location:

appchain-registry/src/registry_settings_actions.rs

- change_minimum_register_deposit
  - value should be greater than 0
- change_counting_interval_in_seconds
  - value should be greater than 0. Possibly bigger than 3600 seconds.

appchain-registry/src/voter_actions.rs

- withdraw_upvote_deposit_of
  - amount should be greater than 0.
- withdraw_downvote_deposit_of
  - amount should be greater than 0.

Risk Level:

Likelihood - 3
Impact - 2

Recommendation:

Consider adding zero value check to those functions.

Remediation Plan:

SOLVED: The Octopus Network team solved the issue by adding necessary zero-value checks.

3.12 (HAL-12) REDUNDANT CODE - LOW

Description:

It was observed that there is a redundant code in `withdraw_upvote_deposit_of` and `withdraw_downvote_deposit_of` in the `voter_actions` module. Even though there is already a variable called `voter = env::predecessor_account_id();` new variable called `account_id = env::predecessor_account_id();` is created and the same actions are performed for both of those variables for `voter_upvote`.

Code Location:

Listing 12: appchain-appchain-registry/src/voter_actions.rs (Lines 28,29,38,41)

```
17 fn withdraw_upvote_deposit_of(&mut self, appchain_id: AppchainId,
       amount: U128) {
18         let voter = env::predecessor_account_id();
19         let voter_upvote = self
20             .upvote_deposits
21             .get(&(appchain_id.clone(), voter.clone()))
22             .unwrap_or_default();
23         assert!(
24             voter_upvote >= amount.0,
25             "Not enough upvote deposit to withdraw."
26         );
27         let mut appchain_basedata = self.get_appchain_basedata(&
               appchain_id);
28         let account_id = env::predecessor_account_id();
29         let voter_upvote = self
30             .upvote_deposits
31             .get(&(appchain_id.clone(), account_id.clone()))
32             .unwrap_or_default();
33         appchain_basedata.decrease_upvote_deposit(amount.0);
34         self.appchain_basedatas
35             .insert(&appchain_id, &appchain_basedata);
36         if amount.0 == voter_upvote {
37             self.upvote_deposits
38                 .remove(&(appchain_id.clone(), account_id.clone())
                        );
39         } else {
40             self.upvote_deposits.insert(
41                 &(appchain_id.clone(), account_id.clone()),
42                 &(voter_upvote - amount.0),
43             );
44         }
45
46
47
```

Risk Level:

Likelihood - 3
Impact - 2

Recommendation:

Consider removing redundant account_id variable and duplicate voter_upvote.

Remediation Plan:

SOLVED: The Octopus Network team solved the issue by removing redundant code.

Fixed Code:

Listing 13: appchain-appchain-registry/src/voter_actions.rs (Lines 33,36)

```
17 fn withdraw_upvote_deposit_of(&mut self, appchain_id: AppchainId,
       amount: U128) {
18         let voter = env::predecessor_account_id();
19         let voter_upvote = self
20             .upvote_deposits
21             .get(&(appchain_id.clone(), voter.clone()))
22             .unwrap_or_default();
23         assert!(
24             voter_upvote >= amount.0,
25             "Not enough upvote deposit to withdraw."
26         );
27         let mut appchain_basedata = self.get_appchain_basedata(&
               appchain_id);
28         appchain_basedata.decrease_upvote_deposit(amount.0);
29         self.appchain_basedatas
30             .insert(&appchain_id, &appchain_basedata);
31         if amount.0 == voter_upvote {
32             self.upvote_deposits
33                 .remove(&(appchain_id.clone(), voter.clone()));
34         } else {
35             self.upvote_deposits.insert(
36                 &(appchain_id.clone(), voter.clone()),
37                 &(voter_upvote - amount.0),
38             );
39         }
```

3.13 (HAL-13) MISSING REASSIGNMENT CHECKS - INFORMATIONAL

Description:

It was observed that the project is missing reassignment checks. Reassignment checks make sure that redundant operations are not performed by not letting the reassignment of the existing value.

Code Location:

appchain-registry/src/registry_settings_actions.rs

- change_operator_of_counting_voting_score: value

appchain-registry/src/lib.rs

- set_owner: owner

Risk Level:

Likelihood - 2
Impact - 1

Recommendation:

Consider adding reassignment checks to avoid performing redundant operations.

Remediation Plan:

SOLVED: The Octopus Network team added all necessary reassignment checks.

3.14 (HAL-14) CODE REFACTOR OPPORTUNITY - INFORMATIONAL

Description:

It was observed that the project is manually restricts the usage of uninitialized smart contract. However, near_sdk already provides a PanicOnDefault macro that generates that code for you.

Code Location:

Listing 14: appchain-registry/src/lib.rs

```
135 impl Default for AppchainRegistry {
136     fn default() -> Self {
137         env::panic(b"The contract needs be initialized before use.
                ")
138     }
139 }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider using PanicOnDefault macro to keep the code cleaner.

Example Code:

Listing 15: appchain-registry/src/lib.rs (Lines 80)

```
79 #[near_bindgen]
80 #[derive(BorshDeserialize, BorshSerialize, PanicOnDefault)]
81 pub struct AppchainRegistry {
82     /// The account of the owner of this contract
83     owner: AccountId,
84     /// The public key of owner account
85     owner_pk: PublicKey,
86     /// The earliest time that the staged code can be deployed
87     contract_code_staging_timestamp: Timestamp,
88     /// The shortest time range between code staging and code
       deployment
89     contract_code_staging_duration: Duration,
90     /// The account of OCT token contract
91     oct_token: AccountId,
92     /// The settings of appchain registry
93     registry_settings: LazyOption<RegistrySettings>,
94     /// The set of all appchain ids
95     appchain_ids: UnorderedSet<AppchainId>,
96     /// The map from appchain id to their basedata
97     appchain_basedatas: LookupMap<AppchainId, AppchainBasedata>,
98     /// The map from pair (appchain id, account id) to their
       upvote deposit
99     upvote_deposits: LookupMap<(AppchainId, AccountId), Balance>,
100    /// The map from pair (appchain id, account id) to their
       downvote deposit
101    downvote_deposits: LookupMap<(AppchainId, AccountId), Balance
       >,
102    /// The appchain id with the highest voting score at a certain
        time
103    top_appchain_id_in_queue: AppchainId,
104    /// The total stake of OCT token in all appchains
105    total_stake: Balance,
106    /// The time of the last calling of function `
       count_voting_score`
107    time_of_last_count_voting_score: Timestamp,
108 }
```

Remediation Plan:

SOLVED: The Octopus Network team solved the issue by adding PanicOnDefault macro.

3.15 (HAL-15) OUTDATED RUST EDITION - INFORMATIONAL

Description:

It was observed that the project is using outdated rust edition(2018). Recently, 2021 rust edition came out, which includes a lot of stability improvements and new features that might make the code more readable.

Code Location:

Listing 16: appchain-registry/Cargo.toml (Lines 5)

```
1 [package]
2 name = "appchain-registry"
3 version = "1.0.5"
4 authors = ["Octopus Network"]
5 edition = "2018"
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider updating the Rust to the latest edition to make use of the latest features and stability improvements.

References:

Rust 2021 Edition Guide

Remediation Plan:

ACKNOWLEDGED: The Octopus Network team acknowledged the issue and is working on fixing it.

AUTOMATED TESTING

4.1 AUTOMATED ANALYSIS

Description:

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-2020-0071	time	Potential segfault in the time crate
RUSTSEC-2021-0110	wasmtime	Multiple Vulnerabilities in Wasmtime

THANK YOU FOR CHOOSING

// HALBORN