Audit Report

Astroport Hub Neutron Migration

v1.0

May 22, 2024

Table of Contents

License

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

THIS AUDIT REPORT WAS PREPARED EXCLUSIVELY FOR AND IN THE INTEREST OF THE CLIENT AND SHALL NOT CONSTRUE ANY LEGAL RELATIONSHIP TOWARDS THIRD PARTIES. IN PARTICULAR, THE AUTHOR AND HIS EMPLOYER UNDERTAKE NO LIABILITY OR RESPONSIBILITY TOWARDS THIRD PARTIES AND PROVIDE NO WARRANTIES REGARDING THE FACTUAL ACCURACY OR COMPLETENESS OF THE AUDIT REPORT.

FOR THE AVOIDANCE OF DOUBT, NOTHING CONTAINED IN THIS AUDIT REPORT SHALL BE CONSTRUED TO IMPOSE ADDITIONAL OBLIGATIONS ON COMPANY, INCLUDING WITHOUT LIMITATION WARRANTIES OR LIABILITIES.

This audit has been performed by

Oak Security GmbH

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security has been engaged by Astroport Protocol Foundation to perform a security audit of Astroport Hub Neutron Migration.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.

2. Determine possible vulnerabilities, which could be exploited by an attacker.

3. Determine smart contract bugs, which might lead to unexpected behavior.

4. Analyze whether best practices have been applied during development.

5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed over multiple phases on the following targets:

Phase 1

Repository	https://github.com/astroport-fi/astroport-governance
Commit	`20c5131c36043f952f27c8d2a02d169a396176c1`
Scope	The scope was restricted to: ● Full audit of `contracts/assembly`. ● Audit of the changes compared to commit `3071dab091f88fac33594574cf3bdb34d9674189` in the following directories: ○ `contract/builder_unlock` ○ `packages/astroport-governance`
Identifier	In this report, all paths pointing to this repository are prefixed with `gov:`
Fixes verified at commit	`75adf9ca8bb72531d541baeaa392591aa45f0ef5` Note that changes to the codebase beyond fixes after the initial audit have not been in the scope of our fixes review.

Repository	https://github.com/astroport-fi/astroport-core
Commit	`4f1b270960cfe5a941e21defe69eded1aa43b562`
Scope	The scope was restricted to the changes compared to commit `cc43f3ed901e8feee334aa7febbb5ac3dfb4ef92`, which includes the following new contracts/packages: ● `contracts/pair_astro_converter` ● `contracts/periphery/astro_converter` ● `contracts/periphery/astro_converter_neutron` ● `packages/astroport/src/astro_converter.rs`
Identifier	In this report, all paths pointing to this repository are prefixed with `core1:`
Fixes verified at commit	`68fc0d3b7263d2d8df435e1c3a7765ef299e9562` Note that changes to the codebase beyond fixes after the initial audit have not been in the scope of our fixes review.

Repository	https://github.com/astroport-fi/astroport-core
Commit	`fa8a81125ef08a5a33fd1b7c2e48cb89112b3977`

Scope	The scope was restricted to the changes compared to commit `cc43f3ed901e8feee334aa7febbb5ac3dfb4ef92`, which includes the following directory and files: ● `contracts/periphery/tokenfactory_tracker` ● `contracts/tokenomics/staking` ● `packages/astroport/src/staking.rs` ● `packages/astroport/src/tokenfactory_tracker.rs` The changes to `contracts/tokenomics/generator` have been excluded from the scope.
Identifier	In this report, all paths pointing to this repository are prefixed with `core2:`
Fixes verified at commit	`57e4766ea0539b9603611060eba1bba051458a42` Note that changes to the codebase beyond fixes after the initial audit have not been in the scope of our fixes review.

Repository	https://github.com/astroport-fi/astroport_ibc
Commit	`fef18175259d86e57b137524eb65bfde28b1e401`
Scope	The scope was restricted to the changes compared to commit `1e70068f8fc3a723bf2a686b6d8f7e1a3b5c427d`, which covers changes to `contracts/cw20-ics20`.
Identifier	In this report, all paths pointing to this repository are prefixed with `ibc:`

Phase 2

Repository	https://github.com/astroport-fi/astroport_ibc
Commit	`d904845ff7da1e513059a158664a6991bf62f134`
Scope	The scope was restricted to the changes compared to commit `fef18175259d86e57b137524eb65bfde28b1e401`.
Identifier	Issues discovered in this commit are detailed in this section.
Fixes verified at commit	`a21339c5173a36218b6b36c2fa76ca744d277cbb` Note that changes to the codebase beyond fixes after the initial audit have not been in the scope of our fixes review.

Phase 3

Repository	https://github.com/astroport-fi/astroport-governance
Commit	`99600ba36669dab00c5b7be23b7c08d58e72e2d3`

Scope	The scope was restricted to changes in only this commit.
Identifier	Issues discovered in this commit are detailed in this section.

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

The Astroport Hub Neutron Migration audit covers the implementation of the migration of Astroport's governance from the Terra to the Neutron blockchain, which includes transitioning ASTRO and xASTRO CW20 tokens into token factory denoms.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending, Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	**Medium**	-
Code readability and clarity	**Medium-High**	Most functions and state variables are documented with clear and concise comments.
Level of documentation	**Medium**	Documentation is available in the `README` files.
Test coverage	**Medium**	The following test coverages are computed with `cargo tarpaulin`: ● `core1`: `80.02%` coverage ● `gov`: `73.04%` coverage ● `ibc`: `52.01%` coverage

Summary of Findings

Phase 1

No	Description	Severity	Status
1	Attackers can manipulate the proposal outcome by transferring voting powers	Critical	Resolved
2	Attackers can bypass self-migration validation to control assembly contract	Critical	Resolved
3	IBC transfers can be grieved, preventing old ASTRO tokens from being burned	Major	Resolved
4	Updating xASTRO denom prevents proposals from being ended	Minor	Resolved
5	Querying total supply for denom validation is inaccurate	Minor	Resolved
6	Tracked denom is not validated	Minor	Resolved
7	Different output when timestamp is provided as `None` and `Some(env.block.time.seconds())`	Minor	Resolved
8	`old_astro_denom` and `outpost_burn_params` validation logic can be improved	Minor	Resolved
9	Lack of events emitted	Informational	Resolved
10	Unused error messages in the codebase	Informational	Resolved
11	`CONFIG` storage state is not exposed through smart queries	Informational	Resolved
12	Inconsistency between comment and implementation	Informational	Resolved
13	Expired proposals remain as `Passed` status	Informational	Resolved

Phase 2

No	Description	Severity	Status
14	`ExecuteFromMultisig` authorization check can be bypassed if the satellite's contract migration admin is the satellite contract itself	Minor	Resolved

Phase 3

No	Description	Severity	Status
15	Centralization concerns from `ExecuteFromMultisig`	Informational	Acknowledged

Detailed Findings

Phase 1

1. Attackers can manipulate the proposal outcome by transferring voting powers

Severity: Critical

When voting on proposals in `gov:contracts/assembly/src/utils.rs:29-32`, the `calc_voting_power` function queries the `BuilderUnlockQueryMsg::Allocation` message to compute the caller's voting power.

An issue occurs when voters transfer their allocations to other voters with the `ProposeNewReceiver` and `ClaimReceiver` messages after voting on a proposal, which also transfers the voting power. This is problematic because new voters can vote on the proposal with the transferred voting power, effectively allowing duplicate voting with the same voting power, inflating total votes, and allowing manipulation of the outcome.

For example, the attacker can submit a malicious proposal and vote with their first address. After transferring the voting power to a second address, the attacker votes again. This step can be repeatedly executed with new addresses until the proposal passes. The impact can be significant as the proposal may contain arbitrary messages, such as transferring all funds or contract migration admin privileges to the attacker's address, potentially causing a loss of funds for other users or the protocol.

Please refer to the `test_manipulate_governance_proposal` test case in the appendix to reproduce the issue.

Recommendation

We recommend implementing `SnapshotMap` on the builder unlock contract so the voting power is determined in the past. The `BuilderUnlockQueryMsg::State` and `BuilderUnlockQueryMsg::Allocation` queries can be updated to include a `timestamp` parameter to compute the voting power at the proposal creation timestamp, similar to the staking contract's implementation in `core2:contracts/periphery/tokenfactory_tracker/src/state.rs:15-21`.

Status: Resolved

2. Attackers can bypass self-migration validation to control assembly contract

Severity: Critical

In gov:contracts/assembly/src/contract.rs:397-403, the check_messages function ensures the supplied Cosmos message does not perform a self-contract migration by validating that the migrated address is not equal to the assembly contract's address. This validation is crucial because unauthorized migrations enable attackers to manipulate the CheckMessagesPassed message to not return an error in line 129, forcing the transaction to succeed and allowing them to control the assembly contract and hence the rest of the protocol.

We found three ways to bypass the validation, as illustrated below.

Firstly, attackers can bypass the existing validation with an uppercase version of the assembly contract's address. Bech32 addresses are case-insensitive, so both uppercase and lowercase addresses resolve to the assembly contract, effectively rendering the validation useless.

Secondly, attackers can use the WasmMsg::UpdateAdmin message to circumvent the validation. The UpdateAdmin message transfers the assembly contract's migration admin to the malicious contract, and the latter WasmMsg::Execute message calls the malicious contract to dispatch a WasmMsg::Migrate message to migrate the assembly contract into the attacker's chosen code ID.

Lastly, attackers can abuse the authz module's MsgGrant to grant the attacker's contract migration permission. This attack can be achieved using CosmosMsg::Stargate to enable migration privilege on the attacker's contract and dispatching MsgExec to trigger the migration, allowing the attacker to take complete control of the assembly contract.

Recommendation

We recommend applying the following recommendations:

- Validate the messages' addresses with the addr_validate function.
- Disallow WasmMsg::UpdateAdmin and MsgGrant messages.

Status: Resolved

3. IBC transfers can be grieved, preventing old ASTRO tokens from being burned

Severity: Major

In core1:contracts/periphery/astro_converter/src/contract.rs:152, the ibc_transfer_for_burning function allows anyone to dispatch an IBC message to transfer the old ASTRO tokens to the Terra chain. Ideally, the client will call the entry point with a rational timeout value so the tokens can be bridged without any issues.

Since this message is permissionless, an attacker can grieve legitimate transactions by front-running the call with a low `timeout` value, causing the transaction to fail due to insufficient funds in the contract. Eventually, the IBC transaction will fail because the timeout will exceed first before sufficient time has passed for the packet to be relayed.

This issue is also present in `core1:contracts/periphery/astro_converter_neutron/src/contract.rs:59`. If the fees are sent to the contract before the `TransferForBurning` message call, the attacker can repeatedly call the function with a low `timeout` value to purposely cause the failure of the IBC transaction, forcing the contract to incur the `timeout_fee` charged by the Neutron chain.

Although there is no profit motive for the attacker, we classify this issue as major because it can cause a loss of fees and disrupt normal protocol operations.

Recommendation

We recommend introducing a contract owner and adding an authorization check for the `TransferForBurning` messages so the caller is the contract owner.

Status: Resolved

4. Updating xASTRO denom prevents proposals from being ended

Severity: Minor

The `assembly` contract allows modifying the `xastro_denom` native token in `gov:contracts/assembly/src/contract.rs:438`. If the native token is updated while there are ongoing proposals, the `end_proposal` function might fail in line `332`, as the new denom will be used instead of the actual denom sent in line `166` and the contract may not hold sufficient funds to execute the proposal.

We classify this issue as minor because the client can recover from it by sending funds of the new denom directly to the contract.

Recommendation

We recommend disallowing the xASTRO denom to be updated.

Status: Resolved

5. Querying total supply for denom validation is inaccurate

Severity: Minor

In `core2:contracts/tokenomics/staking/src/contract.rs:61`, the total supply of `msg.deposit_token_denom` is queried as part of the native token denom validation. If the provided token is not a native token denom, the transaction should revert.

However, this assumption is incorrect because if the token is not a valid denom, the query will not error and instead return a zero value.

Consequently, the contract will fail to work as intended due to an invalid denom, which requires a new deployment from the contract instantiator.

Please refer to the `test_denom_validation` test case in the appendix to reproduce the issue.

Recommendation

We recommend validating that the total supply response is not zero.

Status: Resolved

6. Tracked denom is not validated

Severity: Minor

In `core2:contracts/periphery/tokenfactory_tracker/src/contract.rs:26`, the `tokenfactory_tracker` contract does not validate `tracked_denom` (native token) upon instantiation. If the provided denom is invalid, the sudo messages will fail to work as intended, requiring a new contract deployment from the contract instantiator.

Recommendation

We recommend validating the denom during contract instantiation.

Status: Resolved

7. Different output when timestamp is provided as `None` and `Some(env.block.time.seconds())`

Severity: Minor

In `core2:contracts/tokenomics/staking/src/contract.rs:366-395`, the `BalanceAt` and `TotalSupplyAt` queries allow the caller to specify an optional `timestamp` parameter to query the balance. If the timestamp is provided as `None`, it calls query_balance and query_supply functions that retrieve the balance for the current timestamp. On the other hand, if the timestamp is provided as Some(_), it queries the balance from the tracker contract instead.

However, an inconsistency occurs between the timestamp parameter being provided as None and Some(env.block.time.seconds()). Although both parameters resolve to the current timestamp, the output will be different because None queries the latest value from the bank module. In contrast, Some(env.block.time.seconds()) loads the last recorded value from the SnapshotMap storage, causing the results to differ even though the provided parameter represents the current timestamp.

This may confuse users or third-party applications that use None and Some(env.block.time.seconds()) interchangeably.

Please refer to the test_different_query_results test case in the appendix to reproduce the issue.

Recommendation

We recommend unifying the implementation for cases where the timestamp is provided as None or Some(env.block.time.seconds()).

Status: Resolved

8. old_astro_denom and outpost_burn_params validation logic can be improved

Severity: Minor

In core1:contracts/periphery/astro_converter/src/contract.rs:34, the astro_converter contract sets the old_astro_info and outpost_burn_params parameters upon instantiation. According to the comment in core1:packages/astroport/src/astro_converter.rs:17, the old_astro_denom is expected to be a native IBC-compatible token that uses outpost_burn_params, or a CW20 token that does not require outpost_burn_params to be set. This is crucial to ensure that the old_astro_denom can be burned or bridged correctly as part of the migration from Terra to the Neutron chain.

However, this validation does not cover other configuration possibilities. For example, if old_astro_denom is a CW20 token, the outpost_burn_params must be set to None. Additionally, if the old_astro_info is a native denom but not IBC-compatible, the transaction should revert because the tokens cannot be burned or bridged, which likely means that a configuration mistake has occurred and may require a redeployment.

Recommendation

We recommend raising an error in the following cases:

- `old_astro_denom` is a CW20 token and `outpost_burn_params` is `Some()`.
- `old_astro_denom` is a non-IBC native token denom.

Status: Resolved

9. Lack of events emitted

Severity: Informational

In `gov:contracts/assembly/src/contract.rs:499`, the `update_config` function does not emit information regarding which storage states were updated. It is best practice to emit events and attributes to improve the usability of the contracts and to support off-chain event listeners and blockchain indexers.

Recommendation

We recommend emitting the updated attributes in the `update_config` function.

Status: Resolved

10. Unused error messages in the codebase

Severity: Informational

In several instances across the codebase, multiple `Errors` are defined but not used. This reduces the code readability and maintainability in the codebase.

- `gov:contracts/assembly/src/error.rs`
  - `ProposalNotCompleted`
  - `ProposalNotInDelayPeriod`
  - `MigrationError`
  - `InvalidChannel`
  - `InvalidGeneratorController`
  - `InvalidHub`
  - `InvalidProposalMessages`
  - `InvalidVotingPower`
- `core1:contracts/pair_astro_converter/src/error.rs`
  - `Unauthorized`
  - `InvalidCw20Token`

Recommendation

We recommend implementing the above-mentioned `Errors` in the contract or removing them.

Status: Resolved

11. `CONFIG` storage state is not exposed through smart queries

Severity: Informational

In `core2:contracts/periphery/tokenfactory_tracker/src/query.rs:10`, the `query` entry point does not expose the `CONFIG` storage state value through smart queries. This forces third-party contracts and nodes to perform a raw query to read the stored value, which is error-prone and decreases user experience.

Recommendation

We recommend implementing a smart query that exposes the `CONFIG` storage state.

Status: Resolved

12. Inconsistency between comment and implementation

Severity: Informational

In `gov:packages/astroport-governance/src/assembly.rs:16-17`, the comment mentions that the `DELAY_INTERVAL` constant's range value is between 0.5 to 2 days.

However, the implementation itself ranges from 0.5 to 1 days, which is inconsistent with the comment.

Recommendation

We recommend updating the `DELAY_INTERVAL` constant's range to `16615..=66460` or updating the comment to be 0.5 to 1 day.

Status: Resolved

13. Expired proposals remain as `Passed` status

Severity: Informational

In `gov:contracts/assembly/src/contract.rs:354-356`, the `execute_proposal` function reverts with an `ExecuteProposalExpired` error if the current height exceeds the proposal expiration block. While the logic is correct, the expired proposal will always remain in `Passed` status, which is misleading because passed proposals should be executable.

Recommendation

We recommend updating the status to `ProposalStatus::Expired`.

Status: Resolved

Phase 2

14. `ExecuteFromMultisig` authorization check can be bypassed if the satellite's contract migration admin is the satellite contract itself

Severity: Minor

In `contracts/satellite/src/contract.rs:169-178`, the `exec_from_multisig` function checks that the caller is the satellite contract's migration admin before dispatching Cosmos messages.

Suppose the contract migration admin is the satellite contract itself. In this case, an attacker can bypass the authorization check by dispatching a `CheckMessages` message (see line `157`) to perform a self-call into the `ExecuteFromMultisig` entry point. This bypasses the validation because the caller will be the satellite contract, satisfying the requirement in line `174`.

Consequently, the attacker can migrate the contract into a malicious code ID to prevent erroring in the `CheckMessagesPassed` message, ultimately taking control of the satellite contract.

We classify this issue as minor because it can only be caused if the satellite contract's migration admin is the contract itself (e.g., future development that changes the architecture design). According to the client, the migration admin is the builder's 2 out of 3 multisig contract.

Recommendation

We recommend modifying the `check_messages` function to disallow calling into the satellite contract's `ExecuteFromMultisig` entry point.

Status: Resolved

Phase 3

15. Centralization concerns from `ExecuteFromMultisig`

Severity: Informational

The `assembly` contract includes an `ExecuteFromMultisig` endpoint in `contracts/assembly/src/contract.rs:578-594`, which allows the admin address to execute arbitrary `CosmosMsg` messages through the contract.

This feature allows a centralized entity, or an attacker in case of a compromise, to interact with other contracts of the protocol that implement assembly-only access controls. This allows the entity, for instance, to move any funds held in the target contract.

This issue has been classified as informational as the admin address is supposed to be a 2 out of 3 multisig controlled by the client, which is assumed to be trusted.

Recommendation

We recommend avoiding centralized features. If not possible, the documentation intended for users should clearly state that the organization holds these privileges.

Status: Acknowledged

Appendix

1. Test case for “Attackers can manipulate the proposal outcome by transferring voting powers”

Please run the test case in `gov:contracts/assembly/tests/integration.rs`.

```
#[test]
fn test_manipulate_governance_proposal() {

    use astroport_governance::builder_unlock::msg::ExecuteMsg as
BuilderUnlockExecuteMsg;

    let owner = Addr::unchecked("owner");
    let mut helper = Helper::new(&owner).unwrap();
    let builder_unlock = helper.builder_unlock.clone();

    let user1 = Addr::unchecked("user1");
    let user2 = Addr::unchecked("user2");
    let user3 = Addr::unchecked("user3");
    let user4 = Addr::unchecked("user4");

    // create allocations for user1 and user2
    helper.create_builder_allocation(&user1, 10_000);
    helper.create_builder_allocation(&user2, 10_000);

    // advance block
    helper.next_block(10);

    // create proposal
    helper.get_xastro(&user1, PROPOSAL_REQUIRED_DEPOSIT.u128() + 1000_u128);
    helper.submit_sample_proposal(&user1);

    // user1 votes `yes`
    helper.cast_vote(1, &user1, ProposalVoteOption::For).unwrap();

    // user2 votes `no`
    helper.cast_vote(1, &user2, ProposalVoteOption::Against).unwrap();

    // user1 propose new receiver to user3
    helper.app.execute_contract(user1.clone(), builder_unlock.clone(), &
    BuilderUnlockExecuteMsg::ProposeNewReceiver { new_receiver:
user3.to_string() }, &[]).unwrap();

    // user3 claim allocation
    helper.app.execute_contract(user3.clone(), builder_unlock.clone(), &
    BuilderUnlockExecuteMsg::ClaimReceiver { prev_receiver: user1.to_string() },
&[]).unwrap();
```

```
    // user3 votes `yes`
    helper.cast_vote(1, &user3, ProposalVoteOption::For).unwrap();

    // repeat for user4
    helper.app.execute_contract(user3.clone(), builder_unlock.clone(), &
    BuilderUnlockExecuteMsg::ProposeNewReceiver { new_receiver:
user4.to_string() }, &[]).unwrap();

    helper.app.execute_contract(user4.clone(), builder_unlock.clone(), &
    BuilderUnlockExecuteMsg::ClaimReceiver { prev_receiver: user3.to_string() },
&[]).unwrap();

    helper.cast_vote(1, &user4, ProposalVoteOption::For).unwrap();

    helper.next_block(10);

    let proposal = helper.proposal(1);
    println!("{:?}", proposal);

    assert!(proposal.for_power + proposal.against_power >
proposal.total_voting_power);

    assert!(proposal.for_power > proposal.against_power);
}
```

2. Test case for “Querying total supply for denom validation is inaccurate”

Please run the test case in `core2:contracts/tokenomics/staking/tests/common/helper.rs`.

```
#[test]
fn test_denom_validation() {

       let owner = Addr::unchecked("owner");
       let mut app = BasicAppBuilder::new()
       .with_stargate(StargateKeeper::default())
       .build(|router, _, storage| {
              router
              .bank
              .init_balance(storage, &owner, coins(u128::MAX, ASTRO_DENOM))
              .unwrap()
              });

       let staking_code_id = app.store_code(staking_contract());
       let tracker_code_id = app.store_code(tracker_contract());

       let msg = InstantiateMsg {
       deposit_token_denom: "somethinginvalid".to_string(),
       tracking_admin: owner.to_string(),
       tracking_code_id: tracker_code_id,
       token_factory_addr: TOKEN_FACTORY_MODULE.to_string(),
       };

       app
       .instantiate_contract(
              staking_code_id,
              owner.clone(),
              &msg,
              &[],
              String::from("Astroport Staking"),
              None,
       ).unwrap();

}
```

3. Test case for “Different output when timestamp is provided as None and Some(env.block.time.seconds())”

Please run the test case in core2:contracts/tokenomics/staking/tests/staking_integration.rs.

```
#[test]
fn test_different_query_results() {

        let owner = Addr::unchecked("owner");
        let mut helper = Helper::new(&owner).unwrap();
        let alice = Addr::unchecked("alice");

        // Mint 10000 ASTRO for Alice
        helper.give_astro(10000, &alice);

        // Stake Alice's 1100 ASTRO for 1100 xASTRO
        let resp_data = helper.stake(&alice, 1100).unwrap().data.unwrap();
        let staking_resp: StakingResponse = from_json(&resp_data).unwrap();
        assert_eq!(
        staking_resp,
        StakingResponse {
                astro_amount: 1100u128.into(),
                xastro_amount: 100u128.into(),
        }
        );

        // get current time
        let time_now = helper.app.block_info().time.seconds();

        // query with None, which uses deps.querier.query_balance
        let total_supply_none: Uint128 = helper
        .app
        .wrap()
        .query_wasm_smart(&helper.staking, &QueryMsg::TotalSupplyAt { timestamp:
None })
        .unwrap();

        // query with Some(_), which uses SnapshotMap
        let total_supply_some: Uint128 = helper
        .app
        .wrap()
        .query_wasm_smart(&helper.staking, &QueryMsg::TotalSupplyAt { timestamp:
Some(time_now) })
        .unwrap();

        // underlying Some(time_now) is same as None as both resolves to the
current timestamp, however the query results are diff
        assert_ne!(total_supply_none, total_supply_some);
```

```
       let balance_none: Uint128 = helper
       .app
       .wrap()
       .query_wasm_smart(&helper.staking, &QueryMsg::BalanceAt { timestamp:
None, address: alice.to_string() })
       .unwrap();

       let balance_some: Uint128 = helper
       .app
       .wrap()
       .query_wasm_smart(&helper.staking, &QueryMsg::BalanceAt { timestamp:
Some(time_now), address: alice.to_string() })
       .unwrap();

       assert_ne!(balance_none, balance_some);

}
```