// HALBORN

Astroport.fi AMM Protocol

CosmWasm Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **October 25th, 2021 - November 26th, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	10/25/2021	Luis Quispe Gonzales
0.2	Document Updates	11/16/2021	Piotr Cielas
0.3	Document Updates	11/24/2021	Connor Taylor
0.4	Document Updates	11/26/2021	Luis Quispe Gonzales
0.5	Draft Review	11/30/2021	Gabi Urrutia
1.0	Remediation Plan	12/13/2021	Luis Quispe Gonzales
1.1	Remediation Plan Review	12/14/2021	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
Luis Quispe Gonzales	Halborn	Luis.QuispeGonzales@halborn.com

EXECUTIVE OVERVIEW

1.1 AUDIT SUMMARY

Astroport.fi engaged Halborn to conduct a security assessment on CosmWasm smart contracts beginning on October 25th, 2021 and ending November 26th, 2021.

The security engineers involved on the audit are blockchain and smart contract security experts with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to achieve the following:

- Ensure that smart contract functions work as intended.
- Identify potential security issues with the smart contracts.

In summary, Halborn identified some improvements to reduce the likelihood and impact of risks, which were mostly addressed by Astroport team. The main ones are the following:

- Validate the message sender in maker contract to prevent users from overwriting configuration parameters.
- Harden factory contract to restrict the creation of pools with the same pair or with already existing pairs.
- Handle the case where a pool has no deposits and slippage is specified when users add liquidity.
- Ensure rewards are calculated correctly in generator contract.
- Enforce the use of a default maximum threshold when users add liquidity or swap.
- Enhance owner address transfer functionality.
- Correct the calculus of reverse simulation in stable pairs.

External threats, such as financial related attacks, oracle attacks, and inter-contract functions and calls should be validated for expected logic and state.

1.2 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual review 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 contracts logic related vulnerability.
- Fuzz testing (Halborn custom fuzzing tool)
- Checking the test coverage (cargo tarpaulin)
- Scanning of Rust files for vulnerabilities (cargo audit)

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.

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

1.3 SCOPE

1. CosmWasm Smart Contracts
   (a) Repository: https://github.com/astroport-fi/astroport
   (b) Commit ID: fdc054eb97d758564e123fa1bdc5ae9e7e44e3ae
   (c) Contracts in scope:
      i. contracts/factory
      ii. contracts/pair
      iii. contracts/pair_stable
      iv. contracts/router
      v. contracts/token
      vi. contracts/tokenomics/generator
      vii. contracts/tokenomics/generator_proxy_to_mirror
      viii. contracts/tokenomics/maker
      ix. contracts/tokenomics/staking
      x. contracts/tokenomics/vesting

Out-of-scope: External libraries and financial related attacks

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	3	6	3	7

LIKELIHOOD

IMPACT

	(HAL-05) (HAL-06) (HAL-07) (HAL-08)		(HAL-02)	(HAL-01)
				(HAL-03)
(HAL-13) (HAL-14) (HAL-15)	(HAL-10) (HAL-11)			(HAL-04)
(HAL-16) (HAL-17) (HAL-18) (HAL-19)		(HAL-12)		(HAL-09)

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) ANONYMOUS CONTRACT CONFIG MODIFICATION	High	SOLVED - 11/17/2021
(HAL-02) POSSIBILITY TO CREATE POOLS WITH THE SAME PAIR	High	SOLVED - 11/25/2021
(HAL-03) REPEATED POOLS CAN BE CREATED	High	SOLVED - 12/01/2021
(HAL-04) ADDING LIQUIDITY TO NEW POOLS DOES NOT WORK PROPERLY	Medium	SOLVED - 12/01/2021
(HAL-05) UPDATING A CONFIG PARAMETER AFFECTS PAST REWARDS	Medium	SOLVED - 12/03/2021
(HAL-06) MAXIMUM THRESHOLD FOR SLIPPAGE IS NOT ENFORCED WHEN ADDING LIQUIDITY OR SWAPPING	Medium	SOLVED - 12/08/2021
(HAL-07) PRIVILEGED ADDRESS CAN BE TRANSFERRED WITHOUT CONFIRMATION IN FACTORY	Medium	SOLVED - 12/06/2021
(HAL-08) OWNER ADDRESS NOT TRANSFERABLE	Medium	SOLVED - 12/06/2021
(HAL-09) MISCALCULATION OF REVERSE SIMULATION IN STABLE PAIRS	Medium	SOLVED - 11/17/2021
(HAL-10) ADDRESS VALIDATION MISSING	Low	SOLVED - 11/29/2021
(HAL-11) LACK OF LIMITS WHEN PERFORMING MULTI-SWAP	Low	SOLVED - 12/02/2021
(HAL-12) POSSIBILITY TO CREATE FAKE PAIRS WITH NATIVE COINS	Low	SOLVED - 12/01/2021
(HAL-13) AMOUNT OF TOKENS SENT ARE NOT VALIDATED WHEN EXECUTING SWAP OPERATIONS	Informational	ACKNOWLEDGED
(HAL-14) NO RESTRICTIONS TO DEREGISTER PAIRS	Informational	ACKNOWLEDGED
(HAL-15) MISMATCH OF STABLESWAP FORMULA BETWEEN LITEPAPER AND CONTRACT	Informational	ACKNOWLEDGED

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-16) QUERYING USERS' SHARE IN POOLS COULD PANIC	Informational	SOLVED - 12/01/2021
(HAL-17) OVERFLOW CHECKS NOT SET FOR PROFILE RELEASE	Informational	ACKNOWLEDGED
(HAL-18) INTEGER OVERFLOW	Informational	SOLVED - 12/07/2021
(HAL-19) POSSIBLE MISUSE OF HELPER METHODS	Informational	SOLVED - 12/03/2021

FINDINGS & TECH DETAILS

3.1 (HAL-01) ANONYMOUS CONTRACT CONFIG MODIFICATION - HIGH

Description:

The astroport-maker and astroport-generator contracts hold their configurations in storage. Configuration parameters are set with the set_config, set_allowed_reward_proxies and set_tokens_per_block functions defined in contracts/tokenomics/maker/src/contract.rs and contracts/tokenomics/generator/src/contract.rs. Because those functions do not verify the message sender, malicious users can overwrite configuration parameters.

For example, overwriting the staking and governance contract addresses may redirect new funds to attacker-controlled account(s).

Code Location:

Contract initialisation

Listing 1: tokenomics/maker/src/contract.rs (Lines 30,39)

```
20 #[cfg_attr(not(feature = "library"), entry_point)]
21 pub fn instantiate(
22     deps: DepsMut,
23     _env: Env,
24     info: MessageInfo,
25     msg: InstantiateMsg,
26 ) -> Result<Response, ContractError> {
27     set_contract_version(deps.storage, CONTRACT_NAME,
           CONTRACT_VERSION)?;
28
29     let governance_contract = if let Some(governance_contract) =
           msg.governance_contract {
30         Option::from(deps.api.addr_validate(&governance_contract)
               ?)
31     } else {
32         None
33     };
34
```

```
35      let governance_percent = if let Some(governance_percent) = msg
            .governance_percent {
36          if governance_percent > Uint64::new(100) {
37              return Err(ContractError::IncorrectGovernancePercent
                    {});
38          };
39          governance_percent
40      } else {
41          Uint64::zero()
42      };
```

Listing 2: tokenomics/maker/src/contract.rs (Lines 155,167)

```
147 let governance_amount = if let Some(governance_contract) = cfg.
        governance_contract.clone() {
148     let amount =
149         amount.multiply_ratio(Uint128::from(cfg.governance_percent
                ), Uint128::new(100));
150     let to_governance_asset = Asset {
151         info: info.clone(),
152         amount,
153     };
154     result.push(SubMsg::new(
155         to_governance_asset.into_msg(&deps.querier,
                governance_contract)?,
156     ));
157     amount
158 } else {
159     Uint128::zero()
160 };
161 let staking_amount = amount - governance_amount;
162 let to_staking_asset = Asset {
163     info,
164     amount: staking_amount,
165 };
166 result.push(SubMsg::new(
167     to_staking_asset.into_msg(&deps.querier, cfg.staking_contract.
            clone())?,
168 ));
```

Anonymous config update

Listing 3: tokenomics/maker/src/contract.rs (Lines 70)

```
66 ExecuteMsg::SetConfig {
67     staking_contract,
68     governance_contract,
69     governance_percent,
70 } => set_config(
71     deps,
72     env,
73     staking_contract,
74     governance_contract,
75     governance_percent,
76 ),
```

Listing 4: tokenomics/maker/src/contract.rs (Lines 237,244)

```
225 fn set_config(
226     deps: DepsMut,
227     _env: Env,
228     staking_contract: Option<String>,
229     governance_contract: Option<String>,
230     governance_percent: Option<Uint64>,
231 ) -> Result<Response, ContractError> {
232     let mut event = Event::new("Set config".to_string());
233
234     let mut config = CONFIG.load(deps.storage)?;
235
236     if let Some(staking_contract) = staking_contract {
237         config.staking_contract = deps.api.addr_validate(&
                staking_contract)?;
238         event
239             .attributes
240             .push(Attribute::new("staking_contract", &
                    staking_contract));
241     };
242
243     if let Some(governance_contract) = governance_contract {
244         config.governance_contract = Option::from(deps.api.
                addr_validate(&governance_contract)?);
245         event
246             .attributes
247             .push(Attribute::new("governance_contract", &
```

```
                governance_contract));
248     };
```

Risk Level:

Likelihood - 5
Impact - 4

Recommendations:

Validate the message sender in all three functions function to prevent malicious users from overwriting contract configuration parameters.

Remediation plan:

SOLVED: The issue was fixed in commits 35c6c37a73459166935134e47a6ceb8a1aeff5a1 and 8eede4931d71f7c48c5b469128683b6bd38810f0 -- all three functions validate the sender address now.

3.2 (HAL-02) POSSIBILITY TO CREATE POOLS WITH THE SAME PAIR - HIGH

Description:

`execute_create_pair` function in **contracts/factory/src/contract.rs** allows the possibility to create pools with the same pair, which generates unexpected situations, e.g.: a user could withdraw more tokens than his fair share and affect other users in the pool.

This issue happens because the aforementioned function does not validate if pairs in `asset_infos` have the same value.

A proof of concept video showing how to exploit this security issue is included in the report.

Additionally, it was found that when exploiting this vulnerability with the **ASTRO** token contract, a threat actor could farm additional governance tokens potentially undermining many of the key governance concepts outlined within the Astroport lite paper.

Code Location:

Listing 5: contracts/factory/src/contract.rs

```
187 pub fn execute_create_pair(
188     deps: DepsMut,
189     env: Env,
190     pair_type: PairType,
191     asset_infos: [AssetInfo; 2],
192     init_hook: Option<InitHook>,
193 ) -> Result<Response, ContractError> {
194     let config = CONFIG.load(deps.storage)?;
195 
196     if PAIRS
197         .may_load(deps.storage, &pair_key(&asset_infos))
198         .unwrap_or(None)
199         .is_some()
200     {
```

```
201         return Err(StdError::generic_err("Pair already exists").
                into());
202     }
203
204     // Get pair type from config
205     let pair_config = PAIR_CONFIGS
206         .load(deps.storage, pair_type.to_string())
207         .map_err(|_| ContractError::PairConfigNotFound {})?;
208
209     PAIRS.save(
210         deps.storage,
211         &pair_key(&asset_infos),
212         &PairInfo {
213             liquidity_token: Addr::unchecked(""),
214             contract_addr: Addr::unchecked(""),
215             asset_infos: [asset_infos[0].clone(), asset_infos[1].
                    clone()],
216             pair_type: pair_type.clone(),
217         },
218     )?;
```

Risk Level:

Likelihood - 4
Impact - 4

Recommendations:

Update the logic of `execute_create_pair` function to ensure that pairs in `asset_infos` do not have the same value.

Remediation plan:

SOLVED: The issue was fixed in commit `3e9836de40a05b7b2c94c87f5cd690c1f16a7876`.

3.3 (HAL-03) REPEATED POOLS CAN BE CREATED - HIGH

Description:

execute_create_pair function in **contracts/factory/src/contract.rs** allows the possibility to create pools with already existing pairs. This issue happens because **pair_key** function will consider that two addresses are different if they differ just in their upper / lower cases.

The situation described above can produce the following consequences:

- Potential fixes for HAL-02 can be bypassed and users will be able to create pools with the same pair, which generates unexpected situations, e.g.: a user could withdraw more tokens than his fair share and affect other users in the pool.
- Repeated pools with already existing pairs can severely reduce the liquidity of each pool and, as a consequence, discourage users to add liquidity or swap using Astroport AMM protocol.

A proof of concept video showing how to exploit this security issue is included in the report.

Code Location:

Listing 6: contracts/factory/src/contract.rs (Lines 196,197)

```
187 pub fn execute_create_pair(
188     deps: DepsMut,
189     env: Env,
190     pair_type: PairType,
191     asset_infos: [AssetInfo; 2],
192     init_hook: Option<InitHook>,
193 ) -> Result<Response, ContractError> {
194     let config = CONFIG.load(deps.storage)?;
195
196     if PAIRS
```

```
197             .may_load(deps.storage, &pair_key(&asset_infos))
198             .unwrap_or(None)
199             .is_some()
200     {
201         return Err(StdError::generic_err("Pair already exists").
                    into());
202     }
```

Risk Level:

Likelihood - 5
Impact - 3

Recommendations:

Update the logic of execute_create_pair to turn addresses in asset_infos into lowercase before calling **pair_key** function.

Remediation plan:

SOLVED: The issue was fixed in commits 451dd974e494eefe88301f51732d7cdf09aac3d0 and 55847db04e84bddcf2a4d5607b9f26644c110a3c.

3.4 (HAL-04) ADDING LIQUIDITY TO NEW POOLS DOES NOT WORK PROPERLY - MEDIUM

Description:

When users call provide_liquidity function in **contracts/pair/src/contract.rs** or **contracts/pair_stable/src/contract.rs** to add liquidity to new pools (i.e.: pools with no deposits), the assert_slippage_tolerance function is triggered and will always panic if **slippage** is specified at the beginning of the operation. This situation can produce the following consequences:

- When legitimate users try to add liquidity to new pools, operations will always panic and make users spend transactions fees needlessly.
- To force a new pool to work as expected, a user should transfer tokens directly to the pool without receiving LP tokens in return, and with the risk that another user benefit from his deposit.
- The issues explained above will arise **every time** a new pool is created (or when its deposits become 0) and legitimate users try to add liquidity.

Code Location:

Listing 7: contracts/pair/src/contract.rs (Lines 833,835)

```
832  if Decimal256::from_ratio(deposits[0], deposits[1]) *
         one_minus_slippage_tolerance
833      > Decimal256::from_ratio(pools[0], pools[1])
834      || Decimal256::from_ratio(deposits[1], deposits[0]) *
             one_minus_slippage_tolerance
835          > Decimal256::from_ratio(pools[1], pools[0])
836  {
837      return Err(ContractError::MaxSlippageAssertion {});
838  }
```

Listing 8: contracts/pair_stable/src/contract.rs (Lines 942,944)

```
941  if Decimal256::from_ratio(deposits[0], deposits[1]) *
         one_minus_slippage_tolerance
942      > Decimal256::from_ratio(pools[0], pools[1])
943      || Decimal256::from_ratio(deposits[1], deposits[0]) *
             one_minus_slippage_tolerance
944      > Decimal256::from_ratio(pools[1], pools[0])
945  {
946      return Err(ContractError::MaxSlippageAssertion {});
947  }
```

Risk Level:

Likelihood - 5
Impact - 2

Recommendations:

Update the logic of assert_slippage_tolerance function to handle correctly the case where a pool has no deposits and **slippage** is specified as an argument of the function.

Remediation plan:

SOLVED: The issue was fixed in commit 612e570f16ae0020a9c45fb30c6115dec83850d5.

3.5 (HAL-05) UPDATING A CONFIG PARAMETER AFFECTS PAST REWARDS - MEDIUM

Description:

tokens_per_block is a configuration parameter of the astroport-generator contract. It is used to calculate the amount of rewards a user is eligible for. However, if this parameter's value is modified after pools are created all rewards will be calculated using the updated value, regardless of when deposits were made. For example, if users had deposited tokens when tokens_per_block was x and withdrew when tokens_per_block was y **all** their rewards were calculated using value y.

Code Location:

Listing 9: contracts/generator/src/contract.rs (Lines 43)

```
28 pub fn instantiate(
29     deps: DepsMut,
30     _env: Env,
31     info: MessageInfo,
32     msg: InstantiateMsg,
33 ) -> Result<Response, ContractError> {
34     set_contract_version(deps.storage, CONTRACT_NAME,
           CONTRACT_VERSION)?;
35
36     let mut allowed_reward_proxies: Vec<Addr> = vec![];
37     for proxy in msg.allowed_reward_proxies {
38         allowed_reward_proxies.push(deps.api.addr_validate(&proxy)
               ?);
39     }
40
41     let config = Config {
42         astro_token: deps.api.addr_validate(&msg.astro_token)?,
43         tokens_per_block: msg.tokens_per_block,
44         total_alloc_point: Uint64::from(0u64),
```

Listing 10: tokenomics/generator/src/contract.rs (Lines 758)

```
756 fn set_tokens_per_block(deps: DepsMut, amount: Uint128) -> Result<
      Response, ContractError> {
757     CONFIG.update::<_, ContractError>(deps.storage, |mut v| {
758         v.tokens_per_block = amount;
759         Ok(v)
760     })?;
761     Ok(Response::new()
762         .add_event(Event::new("Set tokens per block").
              add_attribute("amount", amount)))
763 }
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendations:

Call the mass_update_pools function before updating the tokens_per_block parameter to ensure rewards are calculated correctly.

Remediation plan:

SOLVED: The issue was fixed in commit 3a4a8a96e9eec94e8868662eca453601a8d342d9. Pools are now updated before the tokens_per_block parameter's value is changed.

3.6 (HAL-06) MAXIMUM THRESHOLD FOR SLIPPAGE IS NOT ENFORCED WHEN ADDING LIQUIDITY OR SWAPPING - MEDIUM

Description:

When users add liquidity / swap and do not specify slippage tolerance (or its equivalent) in the operation, Astroport AMM protocol does not enforce a default maximum threshold, which could severely affect users' amount of tokens received in return. This issue can produce the following scenarios:

Scenario #1: Adding liquidity

- Someone creates a pool with **8000 token X** and **2000 token Y**, as a consequence, creator receives 4000 LP in return.
- User A sends a transaction to provide liquidity of **80 token X** and **20 token Y** to the pool, so he expects to receive 40 LP in return.
- However, some seconds before transaction of user A is processed, user B swaps **12000 token X** to **1200 token Y**. The final balance in the pool is: **20000 token X** and **800 token Y**.
- When transaction of user A is processed, he receives 16 LP in return, instead of 40 LP he was expecting, i.e.: **less than 50%**.

Scenario #2: Adding liquidity (imbalanced token pair)

If a user mistakenly (or fooled by an attacker) provides liquidity with an imbalanced token pair, he could lose all his excedent tokens. See the following example:

- Someone creates a pool with **8000 token X** and **2000 token Y**, as a consequence, creator receives 4000 LP in return.

- User A provides liquidity of **80 token X** and **20 token Y** to the pool, so he receives 40 LP in return.
- User B provides liquidity of **80 token X** and **2000 token Y**, he also receives 40 LP in return, the same amount of LP tokens than previous transaction, but spending **100 times more** token B.

Scenario #3: Swapping

- Someone creates a pool with **8000 token X** and **2000 token Y**.
- User A sends a transaction to swap **100 token X** and expects to receive ~25 token Y in return.
- However, some seconds before transaction of user A is processed, user B swaps **12000 token X** to **1200 token Y**. The final balance in the pool is: **20000 token X** and **800 token Y**.
- When transaction of user A is processed, he receives ~4 token Y in return, instead of ~25 token Y he was expecting, i.e.: **less than 20%** of expected value.

Some recent DeFi attacks as occurred to BT.Finance or Saddle Finance show the importance to have a maximum predefined slippage to reduce the impact of tokens loss if unexpected situations appear or attackers compromise smart contracts in a platform.

Code Location:

When users add liquidity to a pool, assert_slippage_tolerance function will always return **Ok(())** if slippage is not specified:

Listing 11: contracts/pair/src/contract.rs (Lines 821,841)

```
821 if let Some(slippage_tolerance) = *slippage_tolerance {
822     let slippage_tolerance: Decimal256 = slippage_tolerance.into();
823     if slippage_tolerance > Decimal256::one() {
824         return Err(StdError::generic_err("slippage_tolerance cannot
                bigger than 1").into());
```

```
825     }
826
827     let one_minus_slippage_tolerance = Decimal256::one() -
            slippage_tolerance;
828     let deposits: [Uint256; 2] = [deposits[0].into(), deposits[1].
            into()];
829     let pools: [Uint256; 2] = [pools[0].amount.into(), pools[1].
            amount.into()];
830
831     // Ensure each prices are not dropped as much as slippage
            tolerance rate
832     if Decimal256::from_ratio(deposits[0], deposits[1]) *
            one_minus_slippage_tolerance
833        > Decimal256::from_ratio(pools[0], pools[1])
834        || Decimal256::from_ratio(deposits[1], deposits[0]) *
               one_minus_slippage_tolerance
835           > Decimal256::from_ratio(pools[1], pools[0])
836     {
837         return Err(ContractError::MaxSlippageAssertion {});
838     }
839 }
840
841 Ok(())
```

Listing 12: contracts/pair_stable/src/contract.rs (Lines 930,950)

```
930 if let Some(slippage_tolerance) = *slippage_tolerance {
931     let slippage_tolerance: Decimal256 = slippage_tolerance.into()
            ;
932     if slippage_tolerance > Decimal256::one() {
933         return Err(StdError::generic_err("slippage_tolerance
                cannot bigger than 1").into());
934     }
935
936     let one_minus_slippage_tolerance = Decimal256::one() -
            slippage_tolerance;
937     let deposits: [Uint256; 2] = [deposits[0].into(), deposits[1].
            into()];
938     let pools: [Uint256; 2] = [pools[0].amount.into(), pools[1].
            amount.into()];
939
940     // Ensure each prices are not dropped as much as slippage
            tolerance rate
```

```
941     if Decimal256::from_ratio(deposits[0], deposits[1]) *
            one_minus_slippage_tolerance
942         > Decimal256::from_ratio(pools[0], pools[1])
943         || Decimal256::from_ratio(deposits[1], deposits[0]) *
                one_minus_slippage_tolerance
944             > Decimal256::from_ratio(pools[1], pools[0])
945     {
946         return Err(ContractError::MaxSlippageAssertion {});
947     }
948 }
949
950 Ok(())
```

When users try to swap, assert_max_spread function will always return **Ok(())** if max_spread is not specified:

Listing 13: contracts/pair/src/contract.rs (Lines 795,807,813)

```
788 pub fn assert_max_spread(
789     belief_price: Option<Decimal>,
790     max_spread: Option<Decimal>,
791     offer_amount: Uint128,
792     return_amount: Uint128,
793     spread_amount: Uint128,
794 ) -> Result<(), ContractError> {
795     if let (Some(max_spread), Some(belief_price)) = (max_spread,
            belief_price) {
796         let expected_return =
797             offer_amount * Decimal::from(Decimal256::one() /
                    Decimal256::from(belief_price));
798         let spread_amount = expected_return
799             .checked_sub(return_amount)
800             .unwrap_or_else(|_| Uint128::zero());
801
802         if return_amount < expected_return
803             && Decimal::from_ratio(spread_amount, expected_return)
                    > max_spread
804         {
805             return Err(ContractError::MaxSpreadAssertion {});
806         }
807     } else if let Some(max_spread) = max_spread {
808         if Decimal::from_ratio(spread_amount, return_amount +
                spread_amount) > max_spread {
```

```
809             return Err(ContractError::MaxSpreadAssertion {});
810         }
811     }
812
813     Ok(())
814 }
```

Listing 14: contracts/pair_stable/src/contract.rs (Lines 904,916,922)

```
897 pub fn assert_max_spread(
898     belief_price: Option<Decimal>,
899     max_spread: Option<Decimal>,
900     offer_amount: Uint128,
901     return_amount: Uint128,
902     spread_amount: Uint128,
903 ) -> Result<(), ContractError> {
904     if let (Some(max_spread), Some(belief_price)) = (max_spread,
        belief_price) {
905         let expected_return =
906             offer_amount * Decimal::from(Decimal256::one() /
                Decimal256::from(belief_price));
907         let spread_amount = expected_return
908             .checked_sub(return_amount)
909             .unwrap_or_else(|_| Uint128::zero());
910
911         if return_amount < expected_return
912             && Decimal::from_ratio(spread_amount, expected_return)
                    > max_spread
913         {
914             return Err(ContractError::MaxSpreadAssertion {});
915         }
916     } else if let Some(max_spread) = max_spread {
917         if Decimal::from_ratio(spread_amount, return_amount +
            spread_amount) > max_spread {
918             return Err(ContractError::MaxSpreadAssertion {});
919         }
920     }
921
922     Ok(())
923 }
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendations:

Enforce the use of a default maximum threshold when users add liquidity or swap, but do not specify slippage tolerance (or its equivalent) or slippage value is greater than the threshold. As a reference, **max slippage** for Uniswap Pool and Uniswap Swap is 50%.

Remediation plan:

SOLVED: The issue was fixed in the following commits:

- 163ff75bbd42953eff9669fe2d6d081b7919c3fe
- 52db1fde41737274ec5beb182546ba4f76382752
- fdd6eaec8ce5cb7d5ca7156e4d22ec9654fc7de7

3.7 (HAL-07) PRIVILEGED ADDRESS CAN BE TRANSFERRED WITHOUT CONFIRMATION IN FACTORY - MEDIUM

Description:

An incorrect use of execute_update_config function in **contracts/factory/src/contract.rs** can set owner of **factory** contract to an invalid address and inadvertently lose total control of this contract, which cannot be undone in any way.

Currently, the owner of the **factory** contract can change the **owner address** using the aforementioned function in a single transaction and without confirmation from the new address.

Code Location:

Listing 15: contracts/factory/src/contract.rs (Lines 127)

```
125  if let Some(owner) = owner {
126      // validate address format
127      config.owner = deps.api.addr_validate(owner.as_str())?;
128  }
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendations:

It is recommended to split **owner transfer** functionality into set_owner and accept_ownership functions. The latter function allows the transfer to be completed by the recipient.

Remediation plan:

SOLVED: The issue was fixed in commit d087c04d99f3b44b9c65ebf676fd40cee99d47cf.

3.8 (HAL-08) OWNER ADDRESS NOT TRANSFERABLE - MEDIUM

Description:

Some governance operations on the astroport-generator and astroport-maker contracts require current contract owner signature. Since neither contract implements a governance address transfer function it is impossible to assign a new owner in case the current account is compromised.

Code Location:

Listing 16: tokenomics/generator/src/contract.rs (Lines 45)

```
28 #[cfg_attr(not(feature = "library"), entry_point)]
29 pub fn instantiate(
30     deps: DepsMut,
31     _env: Env,
32     info: MessageInfo,
33     msg: InstantiateMsg,
34 ) -> Result<Response, ContractError> {
35     set_contract_version(deps.storage, CONTRACT_NAME,
           CONTRACT_VERSION)?;
36
37     let mut allowed_reward_proxies: Vec<Addr> = vec![];
38     for proxy in msg.allowed_reward_proxies {
39         allowed_reward_proxies.push(deps.api.addr_validate(&proxy)
               ?);
40     }
41
42     let config = Config {
43         astro_token: deps.api.addr_validate(&msg.astro_token)?,
44         tokens_per_block: msg.tokens_per_block,
45         total_alloc_point: Uint64::from(0u64),
46         owner: info.sender,
```

Listing 17: tokenomics/maker/src/contract.rs (Lines 45)

```
35 let governance_percent = if let Some(governance_percent) = msg.
       governance_percent {
36         if governance_percent > Uint64::new(100) {
37             return Err(ContractError::IncorrectGovernancePercent
                   {});
38         };
39         governance_percent
40     } else {
41         Uint64::zero()
42     };
43
44     let cfg = Config {
45         owner: info.sender,
46         astro_token_contract: deps.api.addr_validate(&msg.
               astro_token_contract)?,
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendations:

Implement governance functions updating the owner addresses in case the current ones are compromised.

Remediation plan:

SOLVED: The issue was fixed in commit bdf6f59e4270303e818b031b619345dfa8d6d19e.

3.9 (HAL-09) MISCALCULATION OF REVERSE SIMULATION IN STABLE PAIRS - MEDIUM

Description:

When users make reverse simulation queries in stable pairs, compute_offer_amount function in **contracts/pair_stable/src/contract.rs** calls **calc_amount** function with an incorrect ask_amount parameter, i.e.: **with commission already deducted**. As a consequence, values returned in reverse simulation queries will **always be lesser** than real ones.

Code Location:

Listing 18: contracts/pair_stable/src/contract.rs (Lines 860)

```
853 let greater_precision = offer_precision.max(ask_precision);
854 let offer_pool = adjust_precision(offer_pool, offer_precision,
        greater_precision)?;
855 let ask_pool = adjust_precision(ask_pool, ask_precision,
        greater_precision)?;
856 let ask_amount = adjust_precision(ask_amount, ask_precision,
        greater_precision)?;
857
858 let offer_amount = adjust_precision(
859     Uint128::new(
860         calc_amount(ask_pool.u128(), offer_pool.u128(),
                ask_amount.u128(), amp).unwrap(),
861     ),
862     greater_precision,
863     offer_precision,
864 )?;
```

Risk Level:

Likelihood - 5

Impact - 1

Recommendations:

Update the logic of compute_offer_amount function to call **calc_amount** function with ask_amount parameter without commission deducted.

Remediation plan:

SOLVED: The issue was fixed in commit 06728064a0eeda2b47dd1f8b1dbe0e975a700ecc. The Astroport team also discovered this security issue while security audit was in progress and solved it timely.

3.10 (HAL-10) ADDRESS VALIDATION MISSING - LOW

Description:

The deposit token address is set on contract initialization from a user-supplied deposit_token_addr parameter. The instantiate function however does not validate its value to match the expected format and it's impossible to update this value after it's initially set.

Code Location:

Listing 19: tokenomics/staking/src/contract.rs (Lines 35)

```
22 #[cfg_attr(not(feature = "library"), entry_point)]
23 pub fn instantiate(
24     deps: DepsMut,
25     env: Env,
26     _info: MessageInfo,
27     msg: InstantiateMsg,
28 ) -> StdResult<Response> {
29     set_contract_version(deps.storage, CONTRACT_NAME,
           CONTRACT_VERSION)?;
30
31     // Store config
32     CONFIG.save(
33         deps.storage,
34         &Config {
35             deposit_token_addr: msg.deposit_token_addr,
36             share_token_addr: Addr::unchecked(""),
37         },
38     )?;
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendations:

Validate the user-supplied parameter value with the DepsMut::api::addr_validate function from the cosmwasm-std crate.

Remediation plan:

SOLVED: The issue was fixed in commit f0bd6798a083096276fd1a9cbf724bfe41f7e49d.

3.11 (HAL-11) LACK OF LIMITS WHEN PERFORMING MULTI-SWAP - LOW

Description:

When performing multi-hop swap operations through the execute_swap_operation function of the router contract, a user may supply multiple tokens to be exchanged to reach a single swap target token. While several validation checks were found to be in place, these checks did not restrict the number of token swaps within an operation.

Due to the serialized nature of CosmWasm contracts, a threat actor may be able to leverage a lack of maximum limits to perform Denial of Service (DoS) attacks against the contract.

Code Location:

Listing 20: router/src/contract.rs (Lines 128,133,136)

```
127     let operations_len = operations.len();
128     if operations_len == 0 {
129         return Err(ContractError::MustProvideOperations {});
130     }
131
132     // Assert the operations are properly set
133     assert_operations(&operations)?;
134
135     let to = if let Some(to) = to { to } else { sender };
136     let target_asset_info = operations.last().unwrap().
            get_target_asset_info();
137
138     let mut operation_index = 0;
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendations:

It is recommended that logic is implemented to limit the number of token swaps within a chain to a reasonable amount within multi-swap operations.

Remediation plan:

SOLVED: The issue was fixed in commit 116709b2b6fbf1abaaf637cf413b438806ca6ef4.

3.12 (HAL-12) POSSIBILITY TO CREATE FAKE PAIRS WITH NATIVE COINS - LOW

Description:

An attacker can create a pool with a pair that contains a fake native token trying to imitate a real one, e.g.: denom in native token uses 'UUSD' / 'Uusd' / 'uUSD' / . . . as value instead of 'uusd', as shown in the following example:

- **Token 1:** { token: { contract_addr: 'terra1...'} }
- **Token 2:** { native_token: { denom: 'UUSD'} }

This pool is created and will appear as a legitimate one in **factory** contract, and when users try to add liquidity to the pool, operations will always fail and make users spend transactions fees needlessly.

This issue happens because pair_key function in **contracts/factory/src/state.rs** does not restrict that denom in native tokens use upper case letters.

Code Location:

Listing 21: contracts/factory/src/state.rs

```
21 pub fn pair_key(asset_infos: &[AssetInfo; 2]) -> Vec<u8> {
22     let mut asset_infos = asset_infos.to_vec();
23     asset_infos.sort_by(|a, b| a.as_bytes().cmp(b.as_bytes()));
24
25     [asset_infos[0].as_bytes(), asset_infos[1].as_bytes()].concat
           ()
26 }
```

Risk Level:

Likelihood - 3
Impact - 1

Recommendations:

Update the logic of `pair_key` function to throw an error message when denom in native tokens use upper case letters.

Remediation plan:

SOLVED: The issue was fixed in commits `451dd974e494eefe88301f51732d7cdf09aac3d0` and `55847db04e84bddcf2a4d5607b9f26644c110a3c`.

3.13 (HAL-13) AMOUNT OF TOKENS SENT ARE NOT VALIDATED WHEN EXECUTING SWAP OPERATIONS - INFORMATIONAL

Description:

When users execute swap operations through ExecuteMsg::ExecuteSwapOperations or Cw20HookMsg::ExecuteSwapOperations messages in **contracts/router/src/contract.rs**, amount of tokens sent in transactions are not validated.

As a consequence, a malicious user could execute swap operations spending other users' tokens stored in the **router** contract. However, this is an unlikely scenario that would only happen if someone mistakenly transfers tokens directly to **router** contract or executes partial swap operations, i.e.: not using all tokens previously transferred.

Code Location:

Listing 22: contracts/router/src/contract.rs

```
55 ExecuteMsg::ExecuteSwapOperations {
56     operations,
57     minimum_receive,
58     to,
59 } => execute_swap_operations(
60     deps,
61     env,
62     info.clone(),
63     info.sender,
64     operations,
65     minimum_receive,
66     to,
67 ),
```

Listing 23: contracts/router/src/contract.rs

```
94 Cw20HookMsg::ExecuteSwapOperations {
95     operations,
96     minimum_receive,
97     to,
98 } => {
99     let to_addr = if let Some(to_addr) = to {
100         Some(deps.api.addr_validate(to_addr.as_str())?)
101     } else {
102         None
103     };
104
105     execute_swap_operations(
106         deps,
107         env,
108         info,
109         sender,
110         operations,
111         minimum_receive,
112         to_addr,
113     )
114 }
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendations:

Update the logic of ExecuteMsg::ExecuteSwapOperations and Cw20HookMsg::ExecuteSwapOperations messages to validate the amount of tokens sent in transactions, using first asset in **SwapOperation** vector as a reference.

Remediation plan::

ACKNOWLEDGED: The Astroport team acknowledged this finding.

3.14 (HAL-14) NO RESTRICTIONS TO DEREGISTER PAIRS - INFORMATIONAL

Description:

deregister function in **contracts/factory/src/contract.rs** allow owner of contract to unrestrictedly deregister pools, i.e.: pair_key value of a pool is not stored in contract's storage anymore.

Most of the time, pools are only deregistered when they migrate to another factory, but this process requires a mass migration mechanism; otherwise, the process could be manual error-prone.

Code Location:

Listing 24: contracts/factory/src/contract.rs

```
297 pub fn deregister(
298     deps: DepsMut,
299     info: MessageInfo,
300     asset_infos: [AssetInfo; 2],
301 ) -> Result<Response, ContractError> {
302     let config = CONFIG.load(deps.storage)?;
303
304     if info.sender != config.owner {
305         return Err(ContractError::Unauthorized {});
306     }
307
308     let pair_info: PairInfo = PAIRS.load(deps.storage, &pair_key(&
            asset_infos))?;
309     PAIRS.remove(deps.storage, &pair_key(&asset_infos));
310
311     Ok(Response::new().add_attributes(vec![
312         attr("action", "deregister"),
313         attr("pair_contract_addr", pair_info.contract_addr),
314     ]))
315 }
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendations:

If not used, it is recommended to remove deregister function in **factory** contract. Otherwise, update the logic of the function to allow a mass migration process of pools to a new factory.

It is also important that this mechanism updates factory address in all pools deployed and has security considerations, e.g.,: restrict address that participate in the migration, use temporary password, etc.

Remediation plan::

ACKNOWLEDGED: The Astroport team acknowledged this finding, also stated that they need an admin-ish remove function to have a possibility to remove some broken or malicious pools from factory and this will be done by governance contract at some point.

3.15 (HAL-15) MISMATCH OF STABLESWAP FORMULA BETWEEN LITEPAPER AND CONTRACT - INFORMATIONAL

Description:

According to **Astroport Litepaper**, the derivation of the formula they use in StableSwap invariant pools can be found on Curve Whitepaper and is represented by the following expression:

$$n^n * A * (x + y) + D = n^n * A * D + \frac{D^{n+1}}{n^n * x * y}$$

Parameters in the formula for StableSwap invariant are the following:

- **n:** Number of different tokens in pool
- **A:** Amplification coefficient (defined by pool creator)
- **x, y:** Pooled tokens
- D: StableSwap invariant

Using n = 2 because **pair_stable** contract uses only 2 different tokens, the formula in Astroport Litepaper is obtained:

$$4 * A * (x + y) + D = 4 * A * D + \frac{D^3}{4 * x * y}$$

However, when y values is calculated in compute_new_balance_out function in **contracts/pair_stable/src/math.rs**, the following formula is used instead (with n = 2), which differs from the original one:

$$2*A*(x+y) + D = 2*A*D + \frac{D^3}{4*x*y}$$

This issue can produce that users receive fewer tokens in return than expected when swapping, see the following example:

1. A pool has a stable pair with **10000 token A** and **10000 token B**.
2. A user tries to swap **5000 token A** and expects to receive ~4984 token B, according to original formula for StableSwap invariant in **Astroport Litepaper**.
3. However, the user will receive ~4968 token B (less than expected) because compute_new_balance_out function in **pair_stable** contract is not using the original formula mentioned above.

Code Location:

Listing 25: contracts/pair_stable/src/math.rs

```
77 fn compute_new_balance_out(leverage: u64, new_source_amount: u128,
       d_val: u128) -> Option<u128> {
78     // Upscale to U256
79     let leverage: U256 = leverage.into();
80     let new_source_amount: U256 = new_source_amount.into();
81     let d_val: U256 = d_val.into();
82
83     // sum' = prod' = x
84     // c =  D ** (n + 1) / (n ** (2 * n) * prod' * A)
85     let c = checked_u8_power(&d_val, N_COINS.checked_add(1)?)?
86         .checked_div(checked_u8_mul(&new_source_amount,
              N_COINS_SQUARED)?.checked_mul(leverage)?)?;
87
88     // b = sum' - (A*n**n - 1) * D / (A * n**n)
89     let b = new_source_amount.checked_add(d_val.checked_div(
          leverage)?)?;
90
91     // Solve for y by approximating: y**2 + b*y = c
92     let mut y_prev: U256;
```

```
93      let mut y = d_val;
94      for _ in 0..ITERATIONS {
95          y_prev = y;
96          y = (checked_u8_power(&y, 2)?.checked_add(c)?)
97              .checked_div(checked_u8_mul(&y, 2)?.checked_add(b)?.
                    checked_sub(d_val)?)?;
98          if y == y_prev {
99              break;
100         }
101     }
102     u128::try_from(y).ok()
103 }
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendations:

The Astroport team stated that the formula used in **pair_stable** contract is the intended one. So, it is recommended to update the formula for the **StableSwap invariant** in Astroport Litepaper to match the calculus in compute_new_balance_out function.

Remediation plan::

ACKNOWLEDGED: The Astroport team acknowledged this finding, also stated that they are using the same formula as **Curve** contracts, which it's an invariant of one described in their whitepaper.

3.16 (HAL-16) QUERYING USERS' SHARE IN POOLS COULD PANIC - INFORMATIONAL

Description:

When users call get_share_in_assets function in **contracts/pair/src/contract.rs** or **contracts/pair_stable/src/contract.rs** to query share in pools, operations could panic if total_share is 0, i.e.: new created pool or when its deposits become 0.

This issue happens because share_ratio is calculated directly and does not handle adequately the case when total_share is 0, e.g.: returning a vector with assets whose amounts are 0.

Code Location:

Listing 26: contracts/pair/src/contract.rs (Lines 407)

```
402 pub fn get_share_in_assets(
403     pools: &[Asset; 2],
404     amount: Uint128,
405     total_share: Uint128,
406 ) -> Vec<Asset> {
407     let share_ratio: Decimal = Decimal::from_ratio(amount,
        total_share);
408
409     pools
410         .iter()
411         .map(|a| Asset {
412             info: a.info.clone(),
413             amount: a.amount * share_ratio,
414         })
415         .collect()
416 }
```

Listing 27: contracts/pair_stable/src/contract.rs (Lines 426)

```
421 pub fn get_share_in_assets(
422     pools: &[Asset; 2],
423     amount: Uint128,
424     total_share: Uint128,
425 ) -> [Asset; 2] {
426     let share_ratio: Decimal = Decimal::from_ratio(amount,
        total_share);
427
428     [
429         Asset {
430             info: pools[0].info.clone(),
431             amount: pools[0].amount * share_ratio,
432         },
433         Asset {
434             info: pools[1].info.clone(),
435             amount: pools[1].amount * share_ratio,
436         },
437     ]
438 }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendations:

Update the logic of get_share_in_assets function to handle correctly the cases where total_share in a pool is 0, i.e.: new created pool or when its deposits become 0.

Remediation plan:

SOLVED: The issue was fixed in commit 07c2a79229e6fdc75dc3a83b296648f003302374.

3.17 (HAL-17) OVERFLOW CHECKS NOT SET FOR PROFILE RELEASE - INFORMATIONAL

Description:

While the overflow-checks parameter is set to **true** in profile.release and implicitly applied to all contracts and packages from in workspace, it is not explicitly enabled in **Cargo.toml** file for each individual contract and package, which could lead to unexpected consequences if the project is refactored.

Code Location:

Listing 28: Resources affected

```
contracts/factory/Cargo.toml
contracts/pair/Cargo.toml
contracts/pair_stable/Cargo.toml
contracts/router/Cargo.toml
contracts/token/Cargo.toml
contracts/tokenomics/generator/Cargo.toml
contracts/tokenomics/generator_proxy_to_mirror/Cargo.toml
contracts/tokenomics/maker/Cargo.toml
contracts/tokenomics/staking/Cargo.toml
contracts/tokenomics/vesting/Cargo.toml
packages/astroport/Cargo.toml
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to explicitly enable overflow checks in each individual contract and package. That measure helps when the project is refactored to prevent unintended consequences.

Remediation plan::

ACKNOWLEDGED: The Astroport team acknowledged this finding.

3.18 (HAL-18) INTEGER OVERFLOW - INFORMATIONAL

Description:

Integer overflows/underflows were identified in the register_vesting_accounts function defined in contracts/tokenomics/vesting/src/contract.rs.

For example, a subtraction underflow may occur when the register_vesting_account function defined in tokenomics/vesting/src/contract.rs is executed.

Integer overflows or underflow occur when the result of an arithmetic operation is outside the possible range for an integer. If the amount exceeds the maximum or is lower than the minimum represented by the number of bits available, it will result in an incorrect value.

The overflow-checks feature is set to true in the top crate and it will cause the contract to panic on any overflow.

Code Location:

Listing 29: tokenomics/staking/src/contract.rs (Lines 119,129,136)

```
118 for sch in &vesting_account.schedules {
119     to_deposit += if let Some(end_point) = &sch.end_point {
120         end_point.amount
121     } else {
122         sch.start_point.amount
123     }
124 }
125
126 if let Some(old_info) = VESTING_INFO.may_load(deps.storage, &
        account_address)? {
127     let mut total = Uint128::zero();
128     for sch in &old_info.schedules {
129         total += if let Some(end_point) = &sch.end_point {
130             end_point.amount
131         } else {
```

```
132                 sch.start_point.amount
133             }
134         }
135
136         to_receive += total - old_info.released_amount;
137     }
```

Listing 30: tokenomics/vesting/src/contract.rs (Lines 151,164)

```
149 match to_deposit.cmp(&to_receive) {
150     Ordering::Greater => {
151         let amount = to_deposit - to_receive;
152         response.messages.push(SubMsg::new(WasmMsg::Execute {
153             contract_addr: config.token_addr.to_string(),
154             funds: vec![],
155             msg: to_binary(&Cw20ExecuteMsg::TransferFrom {
156                 owner: config.owner.to_string(),
157                 recipient: env.contract.address.to_string(),
158                 amount,
159             })?,
160         }));
161         event.attributes.push(attr("deposited", amount));
162     }
163     Ordering::Less => {
164         let amount = to_receive - to_deposit;
165         response.messages.push(SubMsg::new(WasmMsg::Execute {
```

Listing 31: tokenomics/vesting/src/contract.rs (Lines 244,245,252)

```
242 if let Some(end_point) = &sch.end_point {
243     let passed_time =
244         current_time.min(end_point.time).seconds() - sch.
            start_point.time.seconds();
245     let time_period = end_point.time.seconds() - sch.start_point.
        time.seconds();
246     if passed_time != 0 && time_period != 0 {
247         let release_amount_per_second: Decimal = Decimal::
            from_ratio(
248             end_point.amount.checked_sub(sch.start_point.amount)?,
249             time_period,
250         );
251
252         available_amount += Uint128::new(passed_time as u128) *
```

```
                release_amount_per_second;
253     }
254 }
```

Listing 32: tokenomics/generator/src/contract.rs (Lines 453)

```
443 let mut user = USER_INFO
444     .load(deps.storage, (&lp_token, &account))
445     .unwrap_or_default();
446
447 let cfg = CONFIG.load(deps.storage)?;
448 let mut pool = POOL_INFO.load(deps.storage, &lp_token)?;
449
450 update_pool_rewards(deps.branch(), &env, &lp_token, &mut pool, &
      cfg)?;
451
452 if !user.amount.is_zero() {
453     let pending = (user.amount * pool.acc_per_share).checked_sub(
          user.reward_debt)?;
```

Also lines #466, 510, 513, 541, 555, 597 and 600 in tokenomics/generator/src/contract.rs.

Risk Level:

Likelihood - 1
Impact - 1

Recommendations:

Consider using the checked_add, checked_sub or checked_mul methods instead of addition, subtraction, and multiplication operators respectively, to handle overflows gracefully.

Remediation plan:

SOLVED: The issue was fixed in commit 50726743d58fb88bf3e6fbba59ec0158b63f470c.

3.19 (HAL-19) POSSIBLE MISUSE OF HELPER METHODS - INFORMATIONAL

Description:

The intention and use of helper methods in Rust, like unwrap, is very useful for testing environments because a value is forcibly demanded to get an error (aka panic!) if the Option the methods is called on doesn't have Some value or Result. Nevertheless, leaving unwrap functions in production environments is a bad practice. Not only will this cause the program to crash out, or panic!, but also no helpful messages are shown to help the user solve, or understand the reason of the error.

Code Location:

Note: usage of unwrap in Rust tests was excluded from the listing below.

Listing 33

```
auditor@halborn:~/astroport$ grep -nR 'unwrap(' * | grep -v -E '
    example|test'
generator/src/contract.rs:260:                          .map(|v| (
    Addr::unchecked(String::from_utf8(v.0).unwrap()), v.1))
generator/src/contract.rs:300:    Ok(if res.is_none() || !res.
    unwrap().is_zero() {
generator/src/contract.rs:352:                .map(|v| (Addr::
    unchecked(String::from_utf8(v.0).unwrap()), v.1))
maker/src/contract.rs:130:        response.messages.last_mut().
    unwrap().reply_on = ReplyOn::Success;
maker/src/contract.rs:217:                .unwrap(),
maker/src/contract.rs:219:            .unwrap(),
vesting/src/state.rs:42:        .map(|(k, v)| (Addr::unchecked(
    String::from_utf8(k).unwrap()), v))
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendations:

It is recommended not use the unwrap function in production environment because this use provokes panic! and may crash the contract without verbose error messages. Crashing the system will result in a loss of availability, and in some cases, even private information stored in the state. Some alternatives are possible, such as propagating the error with ? instead of unwrap or using the error-chain crate for errors.

Remediation plan:

SOLVED: The issue was fixed in commit b43d0e18de3803044c470779d917b381466aa9e1.

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.

ID	package	Short Description
RUSTSEC-2020-0074	pyo3	Reference counting error in ‘From<Py<T>>‘
RUSTSEC-2021-0003	smallvec	Buffer overflow in SmallVec::insert_many

THANK YOU FOR CHOOSING

// HALBORN