HALBORN

Brokkr Protocol P1 Contracts

CosmWasm Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **March 7th, 2022 - March 23rd, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	03/07/2022	Luis Quispe Gonzales
0.2	Document Updates	03/16/2022	Alexis Fabre
0.3	Document Updates	03/18/2022	Luis Quispe Gonzales
0.4	Draft Version	03/23/2022	Luis Quispe Gonzales
0.5	Draft Review	03/24/2022	Gabi Urrutia
1.0	Remediation Plan	03/27/2022	Luis Quispe Gonzales
1.1	Remediation Plan Review	03/27/2022	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Luis Quispe Gonzales	Halborn	Luis.QuispeGonzales@halborn.com

EXECUTIVE OVERVIEW

1.1 AUDIT SUMMARY

Brokkr engaged Halborn to conduct a security assessment on CosmWasm smart contracts beginning on March 7th, 2022 and ending on March 23rd, 2022.

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 impacts of the risks, which were mostly addressed by the Brokkr team. The main ones are the following:

- Transfer the ownership of contracts to a Governance contract to control changes through voting. Otherwise, enable additional security measures for them.
- When unstaking or claiming BRO rewards, remove stakers' info only if, additionally to the already existing conditions, pending bBRO reward is also zero.
- Consider the block height of each stake when calculating the bBRO rewards.
- Apply one of the proposed on-chain oracle strategies to avoid that TWAP prices become stale.
- Split owner address transfer functionality to allow transfer to be completed by recipient.

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. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.

3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

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

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

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: brotocol-token-contracts
   (b) Commit ID: 6e5b287382d1c3c29d568851ce3038ffff7407a3
   (c) Contracts in scope:
      i. airdrop
      ii. bonding-v1
      iii. oracle
      iv. staking-v1
      v. vesting
      vi. whitelist-sale

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

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	3	2	3	3

LIKELIHOOD

IMPACT

		(HAL-01)		
	(HAL-05)			
	(HAL-06) (HAL-07) (HAL-08)		(HAL-04)	(HAL-02) (HAL-03)
(HAL-09)				
(HAL-11)	(HAL-10)			

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) POSSIBILITY TO TRANSFER AN ARBITRARY AMOUNT OF TOKENS OUT OF SOME CONTRACTS	High	FUTURE RELEASE
(HAL-02) BBRO TOKENS ARE LOST WHEN UNSTAKING OR CLAIMING REWARDS	High	SOLVED - 03/26/2022
(HAL-03) BBRO REWARDS SCHEMA COULD PRODUCE UNFAIR ADVANTAGES	High	PARTIALLY SOLVED
(HAL-04) PRICES FROM TWAP ORACLE CAN BECOME STALE	Medium	SOLVED - 03/23/2022
(HAL-05) PRIVILEGED ADDRESS CAN BE TRANSFERRED WITHOUT CONFIRMATION	Medium	SOLVED - 03/21/2022
(HAL-06) MISSING VALIDATION FOR BASE RATE AND GROWTH PARAMETERS	Low	SOLVED - 03/26/2022
(HAL-07) MISSING VALIDATION FOR BONDING DISCOUNTS AND REWARD RATIO	Low	SOLVED - 03/26/2022
(HAL-08) MISSING VALIDATION FOR MIN AND MAX VALUES OF LOCKUP PERIOD	Low	SOLVED - 03/26/2022
(HAL-09) VESTING SCHEDULES COULD BE UNCLAIMABLE	Informational	ACKNOWLEDGED
(HAL-10) SLIGHT ROUNDING ISSUES WHEN PROVIDING LP TOKENS	Informational	ACKNOWLEDGED
(HAL-11) OVERFLOW CHECKS NOT SET FOR PROFILE RELEASE	Informational	SOLVED - 03/09/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) POSSIBILITY TO TRANSFER AN ARBITRARY AMOUNT OF TOKENS OUT OF SOME CONTRACTS - HIGH

Description:

The **bonding-v1** and **mvp-treasury** contracts allow an admin (Brokkr team) to transfer an arbitrary amount of tokens out of them. Two possible attack scenarios will be described below.

Attack scenario 1:

1. A malicious (or compromised) admin calls the update_config function in the **bonding-v1** contract to change the value of treasury_contract to an address controlled by him.
2. As a consequence of **Step 1**, every time users provide LP tokens or UST to **bonding-v1** contract, they will be transferred to the malicious address.

Attack scenario 2:

1. A malicious (or compromised) admin calls the spend function in **mvp-treasury** contract with recipient = <address_controlled_by_attacker>.
2. As a consequence of **Step 1**, the aforementioned function could fully drain the **mvp-treasury** contract.

Code Location:

A malicious (or compromised) admin could change the value of treasury_contract in **bonding-v1** to an address controlled by him:

Listing 1: contracts/bonding-v1/src/commands.rs (Line 310)

```
305 pub fn update_config(
306     deps: DepsMut,
307     owner: Option<String>,
308     lp_token: Option<String>,
309     rewards_pool_contract: Option<String>,
310     treasury_contract: Option<String>,
```

Listing 2: contracts/bonding-v1/src/commands.rs (Lines 334-336)

```
330     if let Some(rewards_pool_contract) = rewards_pool_contract {
331         config.rewards_pool_contract = deps.api.addr_canonicalize
 ↳ (&rewards_pool_contract)?;
332     }
333
334     if let Some(treasury_contract) = treasury_contract {
335         config.treasury_contract = deps.api.addr_canonicalize(&
 ↳ treasury_contract)?;
336     }
```

spend function allows a malicious (or compromised) admin to fully drain the **mvp-treasury** contract:

Listing 3: contracts/mvp-treasury/src/commands.rs (Lines 23,37)

```
19 pub fn spend(
20     deps: DepsMut,
21     env: Env,
22     asset_info: AssetInfo,
23     recipient: String,
24 ) -> Result<Response, ContractError> {
25     let balance = queries::query_asset_balance(deps.as_ref(), env,
 ↳  asset_info.clone())?.amount;
26     if balance.is_zero() {
27         return Err(ContractError::InsufficientFunds {});
28     }
29
30     let asset = Asset {
31         info: asset_info,
32         amount: balance,
33     };
34
```

```
35      Ok(Response::new()
36          .add_messages(vec![
37              asset.into_msg(&deps.querier, deps.api.addr_validate(&
↳ recipient)?)?
38          ])
39          .add_attributes(vec![("action", "spend")]))
40  }
```

Risk Level:

Likelihood - 3
Impact - 5

Recommendation:

It is recommended to transfer the ownership of contracts mentioned above to a **Governance** contract, which will control the changes proposed through a voting process. Otherwise, the following security measures should be applied:

- Remove the possibility to update the value of treasury_contract in **bonding-v1** contract.
- Update the logic of spend function in **mvp-treasury** contract to verify that the value of recipient belongs to a list of contracts that Brokkr protocol interacts with. This list is defined when **mvp-treasury** is instantiated and should be non-modifiable.
- Brokkr admin should be handled by a multi-sig wallet.

Remediation plan:

PENDING: The Brokkr team stated that they plan to transfer the ownership of the contracts to a full-fledged governance contract setup in the midterm, which will severely reduce the likelihood of liquidity loss exposure attacks. Meanwhile, they will handle the Brokkr admin by a multisig wallet, as suggested in the recommendation.

3.2 (HAL-02) BBRO TOKENS ARE LOST WHEN UNSTAKING OR CLAIMING REWARDS - HIGH

Description:

When a user unstake or claim BRO rewards in **staking-v1** contract, his information is totally removed from STAKERS item if the following conditions are true:

- Total staked amount is zero
- Pending BRO reward is zero

Under the mentioned circumstances, the user won't be able to claim his bBRO rewards, even if the amount is greater than zero, i.e.: he totally loses all his bBRO tokens.

Code Location:

User's information is totally removed from STAKERS item when he unstakes:

Listing 4: contracts/staking-v1/src/commands.rs (Lines 279-280)

```
275 // decrease stake amount
276 state.total_stake_amount = state.total_stake_amount.checked_sub(
 ↳ amount)?;
277 staker_info.unlocked_stake_amount = staker_info.
 ↳ unlocked_stake_amount.checked_sub(amount)?;
278
279 if staker_info.pending_bro_reward.is_zero() && staker_info.
 ↳ total_staked()?.is_zero() {
280     remove_staker_info(deps.storage, &sender_addr_raw);
281 } else {
282     store_staker_info(deps.storage, &sender_addr_raw, &staker_info
 ↳ )?;
283 }
```

User's information is totally removed from STAKERS item when he claims his BRO rewards:

Listing 5: contracts/staking-v1/src/commands.rs (Lines 385-386)

```
382 staker_info.pending_bro_reward = Uint128::zero();
383 staker_info.unlock_expired_lockups(&env.block)?;
384
385 if staker_info.total_staked()?.is_zero() {
386     remove_staker_info(deps.storage, &sender_addr_raw);
387 } else {
388     store_staker_info(deps.storage, &sender_addr_raw, &staker_info
 ↳ )?;
389 }
```

Risk Level:

Likelihood - 5
Impact - 3

Recommendation:

Update the logic of unstake and claim_bro_rewards functions to remove users' information from STAKERS item only if, additionally to the already existing conditions, pending_bbro_reward is also zero.

Remediation plan:

SOLVED: The issue was fixed in commit f9bbc72a85ff872b36691c4992d7a86439a4bba2.

3.3 (HAL-03) BBRO REWARDS SCHEMA COULD PRODUCE UNFAIR ADVANTAGES - HIGH

Description:

When users stake, the compute_normal_bbro_reward function is called to calculate the bBRO rewards accrued. Because of the calculation of bbro_reward value depends on the subtraction between last_distribution_block and last_balance_update, it is possible the following attack scenario:

Attack scenario:

1. Just after a reward distribution, an attacker stakes 10_000000 BRO (Block height: 200).
2. Attacker waits until there is another reward distribution and he front-runs the transaction to additionally stake 80_000000 BRO before the distribution of BRO tokens (Block height: 700).
3. The attacker unstakes 80_000000 BRO just after the reward distribution (could be even in the same block!).
4. The calculation of his bBRO reward will consider that 90_000000 BRO has been staked for 500 blocks (700 - 200). However, only 10_000000 BRO of the total has been staked for 500 blocks. This situation gives the attacker an unfair advantage.

Code Location:

Listing 6: contracts/staking-v1/src/state.rs (Lines 157,163-164,169)

```
155 let stake_amount = self.total_staked()?;
156
157 if stake_amount.is_zero() || state.last_distribution_block < self.
 ↳ last_balance_update {
158     return Ok(());
```

```
159 }
160
161 let epoch_info = query_epoch_info(querier, epoch_manager_contract)
 ↳ ?;
162
163 let epochs_staked = Uint128::from(state.last_distribution_block -
 ↳ self.last_balance_update)
164     .checked_div(Uint128::from(epoch_info.epoch))?;
165
166 let bbro_per_epoch_reward =
167     stake_amount.checked_div(epoch_info.epochs_per_year())? *
 ↳ epoch_info.bbro_emission_rate;
168
169 let bbro_reward = bbro_per_epoch_reward.checked_mul(epochs_staked)
 ↳ ?;
170 self.pending_bbro_reward = self.pending_bbro_reward.checked_add(
 ↳ bbro_reward)?;
171 self.last_balance_update = current_block;
172
173 Ok(())
```

Risk Level:

Likelihood - 5
Impact - 3

Recommendation:

Update the logic of compute_normal_bbro_reward function to consider the block height of each stake when calculating the bBRO rewards.

Remediation plan:

PARTIALLY SOLVED: The Brokkr team claimed that contracts will be launched to mainnet with the following parameters:

- epoch = 17,280 blocks (1 day)
- unstake_period_blocks = 241,920 blocks (14 days)

Because of this setup, the attack vector mentioned above is not profitable unless the rewards are distributed after 14 days, which is highly unlikely, especially with the use of an off-chain trigger (out-of-scope of this audit).

If the above-mentioned parameters are subsequently incorrectly changed with the `update_config` function, this security issue could arise again, so it has been marked as "Partially Solved".

3.4 (HAL-04) PRICES FROM TWAP ORACLE CAN BECOME STALE - MEDIUM

Description:

consult_price function in **contracts/oracle/src/queries.rs** calculates the return value using the price_average stored in **oracle** contract's storage (PRICE_LAST). However, the function does not previously validate if the price has been updated within a reasonable timeframe.

As a consequence, prices calculated in this TWAP oracle can rapidly become stale if users do not bond tokens frequently enough or if Brokkr's off-chain trigger does not work correctly (out-of-scope for this audit), which could affect negatively users' operations or protocol funds.

Code Location:

price_average is extracted from **oracle** contract's storage (PRICE_LAST) without validating if the price has been updated within a reasonable timeframe:

Listing 7: contracts/oracle/src/queries.rs (Lines 47,49,50,52)

```
46 let config = load_config(deps.storage)?;
47 let price_last = load_price_cumulative_last(deps.storage)?;
48
49 let price_average = if config.asset_infos[0].equal(&asset) {
50     price_last.price_0_average
51 } else if config.asset_infos[1].equal(&asset) {
52     price_last.price_1_average
53 } else {
54     return Err(StdError::generic_err("Invalid asset info"));
55 };
```

Return value of consult_price function is calculated using price_average, even if this value is stale:

Listing 8: contracts/oracle/src/queries.rs (Lines 74,78)

```
72 } else {
73     let price_precision = Uint256::from(10_u128.pow(TWAP_PRECISION
↳ .into()));
74     Uint256::from(amount) * price_average / Decimal256::
↳ from_uint256(price_precision)
75 };
76
77 Ok(ConsultPriceResponse {
78     amount: consult_price.into(),
79 })
```

Risk Level:

Likelihood - 4
Impact - 3

Recommendation:

It is recommended to apply one of the following oracle strategies:

- Update the logic of consult_price function to throw an error message if price_average has not been updated within a reasonable timeframe defined in the contract.
- If data freshness is more important for Brokkr protocol, an oracle with moving averages can be used instead, in which the cumulative price variable is measured more often than once per period. See the following reference for more details.

Remediation plan:

SOLVED: The issue was fixed in commit 44a54bddf48a4e28b711449ddcb13d9bf31afdb9.

3.5 (HAL-05) PRIVILEGED ADDRESS CAN BE TRANSFERRED WITHOUT CONFIRMATION - MEDIUM

Description:

An incorrect use of the update_config function in contracts can set owner to an invalid address and inadvertently lose control of the contracts, which cannot be undone in any way. Currently, the owner of the contracts can change **owner address** using the aforementioned function in a single transaction and without confirmation from the new address.

The affected smart contracts are the following:

- airdrop
- bonding-v1
- oracle
- staking-v1
- vesting

Code Location:

Listing 9: contracts/airdrop/src/commands.rs (Lines 24-26)

```
21 pub fn update_config(deps: DepsMut, owner: Option<String>) ->
↳ Result<Response, ContractError> {
22     let mut config = load_config(deps.storage)?;
23
24     if let Some(owner) = owner {
25         config.owner = deps.api.addr_canonicalize(&owner)?;
26     }
```

Listing 10: contracts/bonding-v1/src/commands.rs (Lines 322-324)

```
305 pub fn update_config(
306     deps: DepsMut,
307     owner: Option<String>,
308     lp_token: Option<String>,
309     rewards_pool_contract: Option<String>,
310     treasury_contract: Option<String>,
311     astroport_factory: Option<String>,
312     oracle_contract: Option<String>,
313     ust_bonding_reward_ratio: Option<Decimal>,
314     ust_bonding_discount: Option<Decimal>,
315     lp_bonding_discount: Option<Decimal>,
316     min_bro_payout: Option<Uint128>,
317     vesting_period_blocks: Option<u64>,
318     lp_bonding_enabled: Option<bool>,
319 ) -> Result<Response, ContractError> {
320     let mut config = load_config(deps.storage)?;
321
322     if let Some(owner) = owner {
323         config.owner = deps.api.addr_canonicalize(&owner)?;
324     }
```

Listing 11: contracts/oracle/src/commands.rs (Lines 28-30)

```
21 pub fn update_config(
22     deps: DepsMut,
23     owner: Option<String>,
24     price_update_interval: Option<u64>,
25 ) -> Result<Response, ContractError> {
26     let mut config = load_config(deps.storage)?;
27
28     if let Some(owner) = owner {
29         config.owner = deps.api.addr_canonicalize(&owner)?;
30     }
```

Listing 12: contracts/staking-v1/src/commands.rs (Lines 502-504)

```
488 pub fn update_config(
489     deps: DepsMut,
490     owner: Option<String>,
491     paused: Option<bool>,
492     unstake_period_blocks: Option<u64>,
```

```
493     min_staking_amount: Option<Uint128>,
494     min_lockup_period_epochs: Option<u64>,
495     max_lockup_period_epochs: Option<u64>,
496     base_rate: Option<Decimal>,
497     linear_growth: Option<Decimal>,
498     exponential_growth: Option<Decimal>,
499 ) -> Result<Response, ContractError> {
500     let mut config = load_config(deps.storage)?;
501
502     if let Some(owner) = owner {
503         config.owner = deps.api.addr_canonicalize(&owner)?;
504     }
```

Listing 13: contracts/vesting/src/commands.rs (Lines 71-73)

```
65 pub fn update_config(
66     deps: DepsMut,
67     owner: Option<String>,
68     genesis_time: Option<u64>,
69 ) -> Result<Response, ContractError> {
70     let mut config = load_config(deps.storage)?;
71     if let Some(owner) = owner {
72         config.owner = deps.api.addr_canonicalize(&owner)?;
73     }
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendation:

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 79549c38936e99a89a1fa7aa7e38456032f47389.

3.6 (HAL-06) MISSING VALIDATION FOR BASE RATE AND GROWTH PARAMETERS - LOW

Description:

`instantiate` and `update_config` functions in **staking-v1** contract do not validate that values of `base_rate`, `linear_growth` or `exponential_growth` are less or equal than a **threshold** (e.g.: 0.1) predefined in the contract.

The aforementioned values are used to calculate premium bBRO rewards. If those values are not correctly set, the premium bBRO rewards for users could be much higher than expected.

Code Location:

`instantiate` function does not validate that `base_rate`, `linear_growth` or `exponential_growth` are less or equal than a predefined **threshold**:

Listing 14: contracts/staking-v1/src/contract.rs (Lines 59-61)

```
store_config(
    deps.storage,
    &Config {
        owner: deps.api.addr_canonicalize(&msg.owner)?,
        paused: false,
        bro_token: deps.api.addr_canonicalize(&msg.bro_token)?,
        rewards_pool_contract: deps.api.addr_canonicalize(&msg.
↳ rewards_pool_contract)?,
        bbro_minter_contract: deps.api.addr_canonicalize(&msg.
↳ bbro_minter_contract)?,
        epoch_manager_contract: deps.api.addr_canonicalize(&msg.
↳ epoch_manager_contract)?,
        unstake_period_blocks: msg.unstake_period_blocks,
        min_staking_amount: msg.min_staking_amount,
        lockup_config: LockupConfig {
           min_lockup_period_epochs: msg.min_lockup_period_epochs,
           max_lockup_period_epochs: msg.max_lockup_period_epochs,
           base_rate: msg.base_rate,
```

```
60              linear_growth: msg.linear_growth,
61              exponential_growth: msg.exponential_growth,
62          },
63      },
64  )?;
```

update_config function does not validate that base_rate, linear_growth or exponential_growth are less or equal than a predefined **threshold**:

Listing 15: contracts/staking-v1/src/commands.rs (Lines 527,531,535)

```
526 if let Some(base_rate) = base_rate {
527     config.lockup_config.base_rate = base_rate;
528 }
529
530 if let Some(linear_growth) = linear_growth {
531     config.lockup_config.linear_growth = linear_growth;
532 }
533
534 if let Some(exponential_growth) = exponential_growth {
535     config.lockup_config.exponential_growth = exponential_growth;
536 }
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

Add a validation routine inside instantiate and update_config functions to ensure that values of base_rate, linear_growth and exponential_growth are less or equal than a **threshold** predefined in the contract.

Remediation plan:

SOLVED: The issue was fixed in commit 032d729b4cddd49c990fdb5d9e78c608d21f0d25.

3.7 (HAL-07) MISSING VALIDATION FOR BONDING DISCOUNTS AND REWARD RATIO - LOW

Description:

`instantiate` and `update_config` functions in **bonding-v1** contract do not validate that values of `ust_bonding_reward_ratio`, `ust_bonding_discount` or `lp_bonding_discount` are less or equal than 1.

The aforementioned values are used to calculate rewards distribution and amounts of BRO tokens to claim. If those values are not correctly set, operations will throw error messages and won't allow legitimate users to claim their rewards, thus generating a denial of service (DoS) in Brokkr protocol.

Code Location:

`instantiate` function does not validate that `ust_bonding_discount` or `lp_bonding_discount` are less or equal than 1:

Listing 16: contracts/bonding-v1/src/contract.rs (Lines 63-64)

```
46 if msg.ust_bonding_reward_ratio > Decimal::from_str("1.0")?
47     || msg.ust_bonding_reward_ratio <= Decimal::zero()
48 {
49     return Err(ContractError::InvalidUstBondRatio {});
50 }
51
52 store_config(
53     deps.storage,
54     &Config {
55         owner: deps.api.addr_canonicalize(&msg.owner)?,
56         bro_token: deps.api.addr_canonicalize(&msg.bro_token)?,
57         lp_token: deps.api.addr_canonicalize(&msg.lp_token)?,
58         rewards_pool_contract: deps.api.addr_canonicalize(&msg.
↳ rewards_pool_contract)?,
```

```
59          treasury_contract: deps.api.addr_canonicalize(&msg.
↳ treasury_contract)?,
60          astroport_factory: deps.api.addr_canonicalize(&msg.
↳ astroport_factory)?,
61          oracle_contract: deps.api.addr_canonicalize(&msg.
↳ oracle_contract)?,
62          ust_bonding_reward_ratio: msg.ust_bonding_reward_ratio,
63          ust_bonding_discount: msg.ust_bonding_discount,
64          lp_bonding_discount: msg.lp_bonding_discount,
65          min_bro_payout: msg.min_bro_payout,
66          vesting_period_blocks: msg.vesting_period_blocks,
67          lp_bonding_enabled: msg.lp_bonding_enabled,
68      },
69  )?;
```

update_config function does not validate that ust_bonding_reward_ratio, ust_bonding_discount or lp_bonding_discount are less or equal than 1:

Listing 17: contracts/bonding-v1/src/commands.rs (Lines 347,351,355)

```
346 if let Some(ust_bonding_reward_ratio) = ust_bonding_reward_ratio {
347     config.ust_bonding_reward_ratio = ust_bonding_reward_ratio;
348 }
349
350 if let Some(ust_bonding_discount) = ust_bonding_discount {
351     config.ust_bonding_discount = ust_bonding_discount;
352 }
353
354 if let Some(lp_bonding_discount) = lp_bonding_discount {
355     config.lp_bonding_discount = lp_bonding_discount;
356 }
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

Add a validation routine inside `instantiate` and `update_config` functions to ensure that values of `ust_bonding_reward_ratio`, `ust_bonding_discount` and `lp_bonding_discount` are less or equal than 1.

Remediation plan:

SOLVED: The issue was fixed in commit `e80b7dc97ff20b683cd27d7a4cdaa6d7d60c1076`.

3.8 (HAL-08) MISSING VALIDATION FOR MIN AND MAX VALUES OF LOCKUP PERIOD - LOW

Description:

`instantiate` and `update_config` functions in **staking-v1** contract do not validate that `min_lockup_period_epochs` is less than `max_lockup_period_epochs`.

The aforementioned values are used to validate the lockup period when staking locked BRO tokens or locking a previous staked amount. If those values are not correctly set, operations will throw error messages and won't allow legitimate users to stake or lock BRO tokens, thus generating a denial of service (DoS) in Brokkr protocol.

Code Location:

`instantiate` function does not validate that `min_lockup_period_epochs` is less than `max_lockup_period_epochs`:

Listing 18: contracts/staking-v1/src/contract.rs (Lines 57-58)

```
store_config(
    deps.storage,
    &Config {
        owner: deps.api.addr_canonicalize(&msg.owner)?,
        paused: false,
        bro_token: deps.api.addr_canonicalize(&msg.bro_token)?,
        rewards_pool_contract: deps.api.addr_canonicalize(&msg.
↳ rewards_pool_contract)?,
        bbro_minter_contract: deps.api.addr_canonicalize(&msg.
↳ bbro_minter_contract)?,
        epoch_manager_contract: deps.api.addr_canonicalize(&msg.
↳ epoch_manager_contract)?,
        unstake_period_blocks: msg.unstake_period_blocks,
        min_staking_amount: msg.min_staking_amount,
        lockup_config: LockupConfig {
```

```
            min_lockup_period_epochs: msg.min_lockup_period_epochs,
            max_lockup_period_epochs: msg.max_lockup_period_epochs,
            base_rate: msg.base_rate,
            linear_growth: msg.linear_growth,
            exponential_growth: msg.exponential_growth,
        },
    },
)?;
```

update_config function does not validate that min_lockup_period_epochs is less than max_lockup_period_epochs:

Listing 19: contracts/staking-v1/src/commands.rs (Lines 519,523)

```
518 if let Some(min_lockup_period_epochs) = min_lockup_period_epochs {
519     config.lockup_config.min_lockup_period_epochs =
 ↳ min_lockup_period_epochs;
520 }
521
522 if let Some(max_lockup_period_epochs) = max_lockup_period_epochs {
523     config.lockup_config.max_lockup_period_epochs =
 ↳ max_lockup_period_epochs;
524 }
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

Add a validation routine inside instantiate and update_config functions to ensure that min_lockup_period_epochs is less than max_lockup_period_epochs.

Remediation plan:

SOLVED: The issue was fixed in commit b9c1e4ad60fa79e030737e5374a8b027c147d091.

3.9 (HAL-09) VESTING SCHEDULES COULD BE UNCLAIMABLE - INFORMATIONAL

Description:

When registering new vesting accounts, the vesting contract sets the last claimed time to the configured genesis time. That parameter could be greater than the expiration date of some vesting schedules, such that `vesting_info.last_claim_time > vesting_info.end_time`, which makes it unclaimable. However, it is possible for the administrator to update genesis time and set new vesting schedules.

Code Location:

Listing 20: packages/services/src/vesting.rs, (Line 101)

```
101 if current_time > schedule.end_time && self.last_claim_time <
 ↳ schedule.end_time {
102     claimable_amount += schedule.bro_amount;
103 }
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

When registering vesting schedules, it is recommended to verify that the end time of that schedule is greater than the genesis time.

Remediation plan:

ACKNOWLEDGED: The Brokkr team acknowledged this finding.

3.10 (HAL-10) SLIGHT ROUNDING ISSUES WHEN PROVIDING LP TOKENS - INFORMATIONAL

Description:

When calculating the **return values** in get_share_in_assets function every time users provide LP tokens to **bonding-v1** contract, the "multiply before divide" principle is not followed, which could generate slight rounding issues in some edge scenarios.

Code Location:

Listing 21: contracts/bonding-v1/src/commands.rs (Lines 412,414)

```
404 fn get_share_in_assets(
405     querier: &QuerierWrapper,
406     bro_pool: &Asset,
407     ust_pool: &Asset,
408     lp_amount: Uint128,
409     lp_token_addr: Addr,
410 ) -> StdResult<(Uint128, Uint128)> {
411     let total_share = query_supply(querier, lp_token_addr)?;
412     let share_ratio = Decimal::from_ratio(lp_amount, total_share);
413
414     Ok((bro_pool.amount * share_ratio, ust_pool.amount *
 ↳ share_ratio))
415 }
```

Risk Level:

Likelihood - 2
Impact - 1

Recommendation:

Return values in get_share_in_assets function should be calculated following the “multiply before divide” principle to reduce rounding issues.

Remediation plan:

ACKNOWLEDGED: The Brokkr team acknowledged this finding.

3.11 (HAL-11) OVERFLOW CHECKS NOT SET FOR PROFILE RELEASE - INFORMATIONAL

Description:

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

Code Location:

Listing 22: Cargo.toml file from root workspace

```
1 [workspace]
2 members = ["packages/*", "contracts/*"]
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to enable overflow checks explicitly in **Cargo.toml** file from root workspace. That measure helps when the project is refactored to prevent unintended consequences.

Remediation plan:

SOLVED: The issue was fixed in commit 2cecb7ba9a75b642883f009f360a28a9551e07ff. The Brokkr team also discovered this security issue while security audit was ongoing and solved it on time.

THANK YOU FOR CHOOSING

// HALBORN