Brokkr Protocol P3 Contracts

CosmWasm Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **March 28th, 2022 - April 1st, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	03/28/2022	Alexis Fabre
0.2	Document Updates	03/31/2022	Alexis Fabre
0.3	Draft Version	04/01/2022	Alexis Fabre
0.4	Draft Review	04/01/2022	Gabi Urrutia
1.0	Remediation Plan	04/11/2022	Alexis Fabre
1.1	Remediation Plan Review	04/13/2022	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
Alexis Fabre	Halborn	Alexis.Fabre@halborn.com

EXECUTIVE OVERVIEW

1.1 AUDIT SUMMARY

Brokkr engaged Halborn to conduct a security assessment on CosmWasm smart contracts beginning on March 28th, 2022 and ending on April 1st, 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 addressed by the Brokkr team. The main ones are the following:

- Split owner address transfer functionality to allow transfer to be completed by recipient.
- Ensure proper verification when checking for spend limits.
- Apply validation mechanisms when updating the configurations.

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.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

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

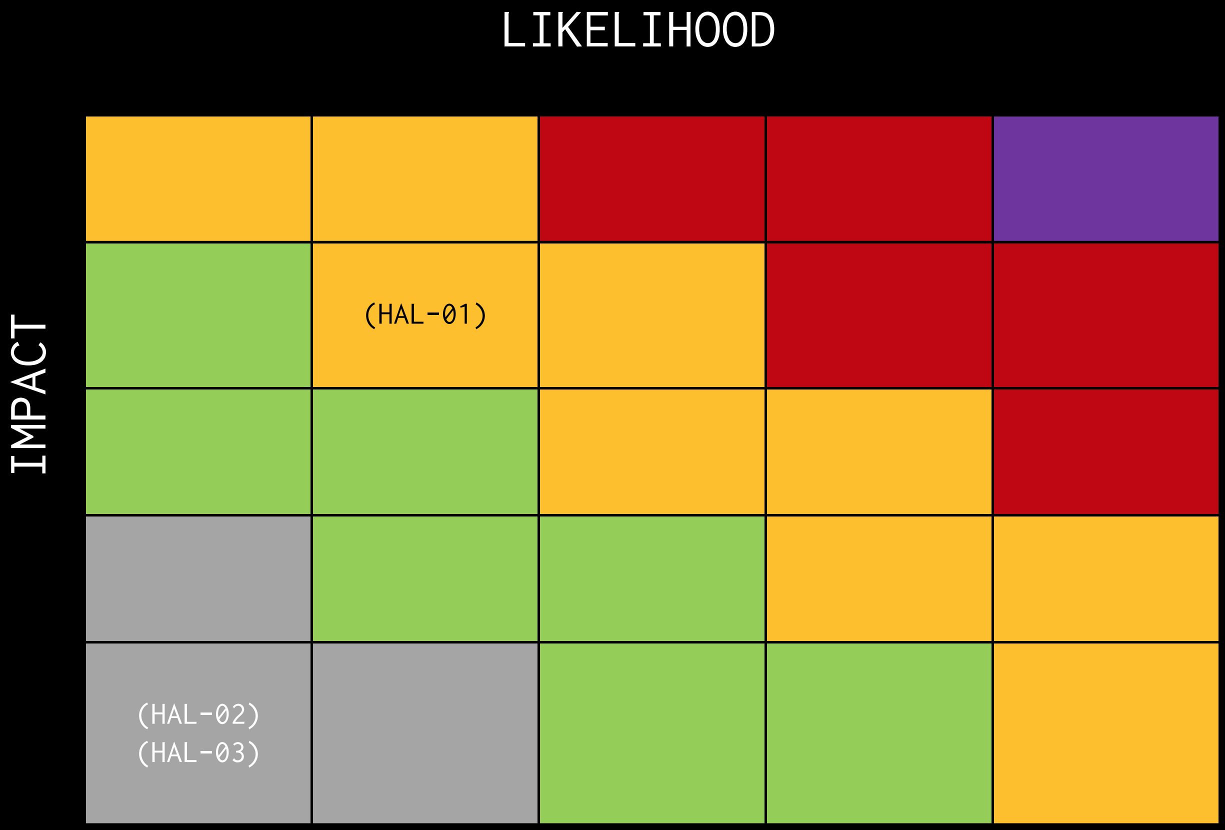
1.3 SCOPE

1. CosmWasm Smart Contracts
   (a) Repository: brotocol-token-contracts
   (b) Commit ID: 6e5b287382d1c3c29d568851ce3038ffff7407a3
   (c) Contracts in scope:
      i. epoch manager
      ii. mvp-tresury
      iii. rewards
      iv. token-pool

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

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	0	2



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) PRIVILEGED ADDRESS CAN BE TRANSFERRED WITHOUT CONFIRMATION	Medium	SOLVED - 03/21/2022
(HAL-02) REWARDS SPEND LIMIT CAN BE IGNORED	Informational	SOLVED - 04/08/2022
(HAL-03) LACK OF VALIDATION WHEN UPDATING EPOCH STATE	Informational	SOLVED - 04/08/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) PRIVILEGED ADDRESS CAN BE TRANSFERRED WITHOUT CONFIRMATION - MEDIUM

Description:

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

The affected smart contracts are the following:

- rewards
- epoch-manager
- token-pool

Code Location:

In the epoch manager contract:

Listing 1: contracts/epoch-manager/src/commands.rs (Line 20)

```
if let Some(owner) = owner {
    config.owner = deps.api.addr_canonicalize(&owner)?;
}
```

In the rewards contract:

Listing 2: contracts/rewards/src/commands.rs (Line 30)

```
if let Some(owner) = owner {
    config.owner = deps.api.addr_canonicalize(&owner)?;
}
```

In the token-pool contract:

Listing 3: contracts/rewards/src/commands.rs (Line 93)

```
if let Some(owner) = owner {
    config.owner = deps.api.addr_canonicalize(&owner)?;
}
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendation:

It is recommended to split **owner transfer** functionality into the set_owner and accept_ownership functions. The last function allows the recipient to complete the transfer.

Remediation plan:

SOLVED: The issue was fixed in commit 79549c38936e99a89a1fa7aa7e38456032f47389.

3.2 (HAL-02) REWARDS SPEND LIMIT CAN BE IGNORED - INFORMATIONAL

Description:

The rewards contract has a spend_limit attribute that caps the maximum amount a distributor can spend in a transaction. However, the amount of each distribution message is verified separately instead of checking that the global amount spent is less than the spending limit. That allows a distributor to bypass the restriction in a transaction.

Users who interact with the distribute function must be whitelisted first. Therefore, the likelihood of such a scenario is low.

Code Location:

Listing 4: contracts/rewards/src/commands.rs, (Lines 129-131)

```rust
let mut msgs: Vec<CosmosMsg> = vec![];
for distribution in distributions {
    if config.spend_limit < distribution.amount {
        return Err(ContractError::SpendLimitReached {});
    }

    msgs.push(CosmosMsg::Wasm(WasmMsg::Execute {
        contract_addr: bro_token.clone(),
        funds: vec![],
        msg: to_binary(&Cw20ExecuteMsg::Send {
            contract: distribution.contract,
            amount: distribution.amount,
            msg: distribution.msg,
        })?,
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to compare the total amount spent with the spend_limit.

Remediation plan:

SOLVED: The issue was fixed in commit a9856345f269ca5275236297feae192d1ee4cec4.

3.3 (HAL-03) LACK OF VALIDATION WHEN UPDATING EPOCH STATE - INFORMATIONAL

Description:

When updating the epoch manager status, the owner can set the following attributes:

- epochs (number of blocks in an epoch)
- blocks_per_year
- bbro_emission_rate

The absence of validation allows the variables to be set to 0, inducing malfunction in the contracts by querying the **epoch-manager**: **staking-v1** and **distributor-v1**. For example, epochs with a value of 0 will cause a 0-unverified division in the distributor-v1 contract, and a 0-verified division in the **staking-v1** contract, causing transactions to go into panic.

However, only an administrator can update these values. Therefore, the likelihood is limited.

Code Location:

Absence of validation when updating the state value in epoch-manager:

Listing 5: contracts/epoch-manager/src/commands.rs, (Line 60)

```
let mut state = load_state(deps.storage)?;

if let Some(epoch) = epoch {
    state.epoch = epoch;
}

if let Some(blocks_per_year) = blocks_per_year {
    state.blocks_per_year = blocks_per_year;
}

if let Some(bbro_emission_rate) = bbro_emission_rate {
    state.bbro_emission_rate = bbro_emission_rate;
}

store_state(deps.storage, &state)?;
```

In staking-v1, when calling compute_normal_bbro_reward:

Listing 6: contracts/staking-v1/src/state.rs, (Line 164)

```
let epoch_info = query_epoch_info(querier, epoch_manager_contract)
↳ ?;

let epochs_staked = Uint128::from(state.last_distribution_block -
↳ self.last_balance_update)
    .checked_div(Uint128::from(epoch_info.epoch))?;
```

In distributor, when calling distribute:

Listing 7: contracts/distributor-v1/src/commands.rs, (Line 44)

```
// query epoch from epoch_manager contract
let epoch_blocks = query_epoch_info(
    &deps.querier,
    deps.api.addr_humanize(&config.epoch_manager_contract)?,
)?
.epoch;

// distribute rewards only for passed epochs
let blocks_since_last_distribution = env.block.height - state.
↳ last_distribution_block;
let passed_epochs = blocks_since_last_distribution / epoch_blocks;
if passed_epochs == 0 {
    return Err(ContractError::NoRewards {});
}
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to check the values of the status parameters before storing them.

Remediation plan:

SOLVED: The issue was fixed in commit 1c7ab7d56ae6ad16c88d8fdfeddb4e3e8e571f85.

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-2020-0025	bigint	bigint is unmaintained, use uint instead

THANK YOU FOR CHOOSING

// HALBORN