Brokkr Protocol P2 Contracts

CosmWasm Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **March 21st, 2022 - March 30th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	03/21/2022	Alexis Fabre
0.2	Document Updates	03/28/2022	Alexis Fabre
0.3	Draft Version	03/30/2022	Alexis Fabre
0.4	Draft Review	03/30/2022	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 21st, 2022 and ending on March 30th, 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:

- Split owner address transfer functionality to allow transfer to be completed by recipient.
- Use checked arithmetical operations.

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

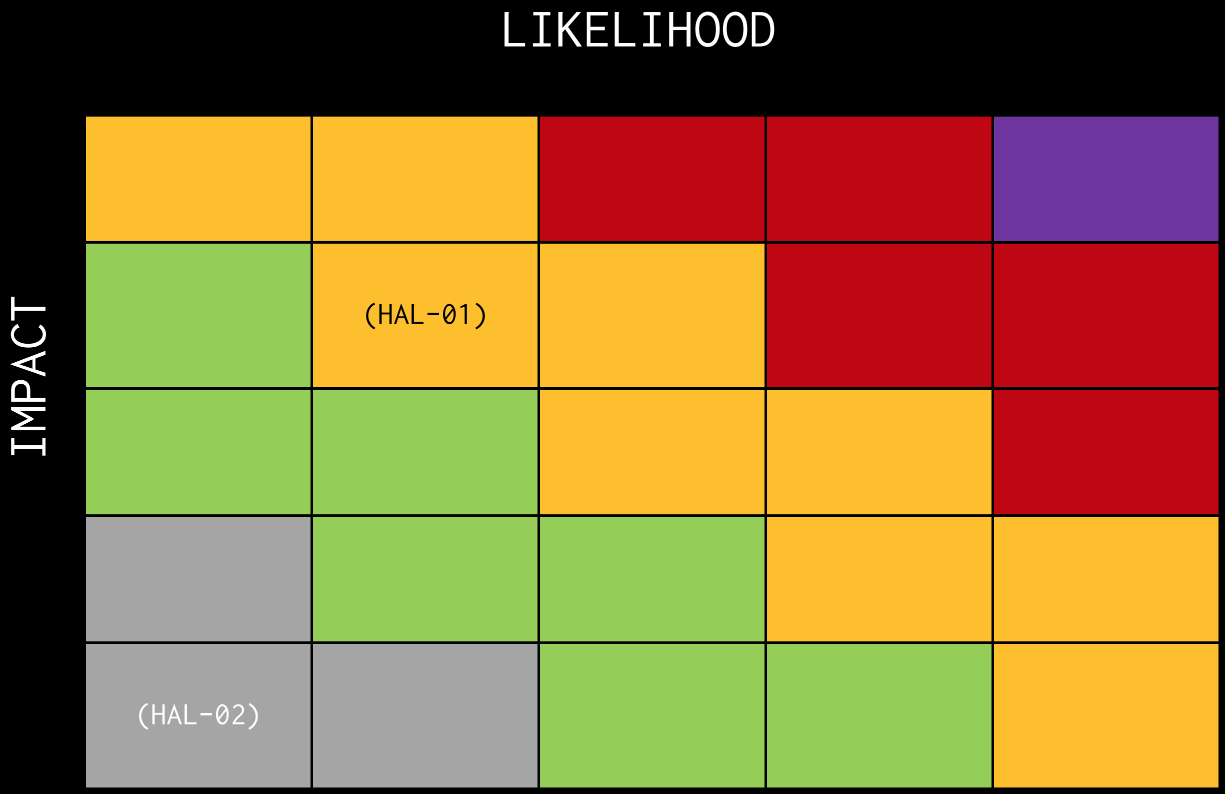
1.3 SCOPE

1. CosmWasm Smart Contracts
   (a) Repository: brotocol-token-contracts
   (b) Commit ID: 6e5b287382d1c3c29d568851ce3038ffff7407a3
   (c) Contracts in scope:
      i. bbro-token
      ii. bbro-minter
      iii. distributor-v1

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

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	0	1



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) PRIVILEGED ADDRESS CAN BE TRANSFERRED WITHOUT CONFIRMATION	Medium	SOLVED - 03/21/2022
(HAL-02) UNCHECKED ARITHMETICAL OPERATIONS CAN CAUSE PANIC	Informational	ACKNOWLEDGED

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 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:

- epoch-manager
- distributor-v1

Code Location:

Listing 1: contracts/distributor-v1/src/commands.rs (Lines 23-26)

```
let mut config = load_config(deps.storage)?;

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

Listing 2: contracts/bbro-minter/src/commands.rs (Lines 28-30)

```
let mut config = load_config(deps.storage)?;

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

Risk Level:

Likelihood - 2
Impact - 4

Recommendation:

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

Remediation plan:

SOLVED: The issue was fixed in commit 79549c38936e99a89a1fa7aa7e38456032f47389.

3.2 (HAL-02) UNCHECKED ARITHMETICAL OPERATIONS CAN CAUSE PANIC - INFORMATIONAL

Description:

When calculating the distribution of rewards, the distributor-v1 contract performs a division without checking that the denominator is not zero. In the event that epoch_manager is misconfigured and has an epoch length of 0 blocks (that can happen because the epoch manager does not check for that condition), the distribute function would panic, leaving the user without a precise error message for your failed transaction.

Code Location:

Listing 3: contracts/distributor-v1/src/commands.rs, (Line 44)

```
35 // query epoch from epoch_manager contract
36 let epoch_blocks = query_epoch_info(
37     &deps.querier,
38     deps.api.addr_humanize(&config.epoch_manager_contract)?,
39 )?
40 .epoch;
41
42 // distribute rewards only for passed epochs
43 let blocks_since_last_distribution = env.block.height - state.
 ↳ last_distribution_block;
44 let passed_epochs = blocks_since_last_distribution / epoch_blocks;
45 if passed_epochs == 0 {
46     return Err(ContractError::NoRewards {});
47 }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

When performing arithmetic operations, it is recommended that you use checked arithmetic. For example, the line of code could be replaced with: blocks_since_last_distribution.checked_div(epoch_blocks).

Remediation plan:

ACKNOWLEDGED: The Brokkr team acknowledged this finding. They also stated that since epoch_blocks is a primitive type, there is no need to convert it to a wrapped as that would break the whole protocol.

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 the vulnerabilities shown here were already disclosed in the previous report. However, to better help developers maintain this code, the auditors include the output with the dependency tree, and this is included in the cargo audit output to better understand 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