HALBORN

Brokkr Protocol Long-Term Bonding

CosmWasm Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **April 19th, 2022 - April 22nd, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	04/19/2022	Jakub Heba
0.2	Document Updates	04/22/2022	Jakub Heba
0.3	Draft Version	04/28/2022	Jakub Heba
0.4	Draft Review	04/29/2022	Gabi Urrutia
1.0	Remediation Plan	05/09/2022	Jakub Heba
1.1	Remediation Plan Review	05/10/2022	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
Jakub Heba	Halborn	Jakub.Heba@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Brokkr Protocol engaged Halborn to conduct a security audit on their smart contracts beginning on April 19th, 2022 and ending on April 22nd, 2022. The security assessment was scoped to the staking-v1 and bonding-v1 smart contracts and limited to the newly created Long-Term Bonding functionality. Code was provided in the GitHub repository, using Git tag Brotocol Token Contracts. Further details can be found in the Scope section of this report.

1.2 AUDIT SUMMARY

The team at Halborn was provided four days for the engagement and assigned a full-time security engineer to audit the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

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

In summary, Halborn identified some improvement to reduce the likelihood and impact of risks, which were acknowledged by Brokkr team. The main ones are the following:

- Normalize all addresses provided by users in contract's logic.
- Use only up-to-date dependencies with no publicly known security issues.

1.3 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 the architecture, purpose, and use of the platform.
- Smart contract manual code review and walkthrough to identify any logic issue.
- Thorough assessment of safety and usage of critical Rust variables and functions in scope that could led to arithmetic vulnerabilities.
- Finding unsafe Rust code usage (cargo-geiger)
- Active Fuzz testing (Halborn custom fuzzing tool).
- Test coverage review (cargo tarpaulin).
- Scanning of Rust files for common vulnerabilities (cargo audit).
- Local or public Testnet deployment (LocalTerra or bombay-12)

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

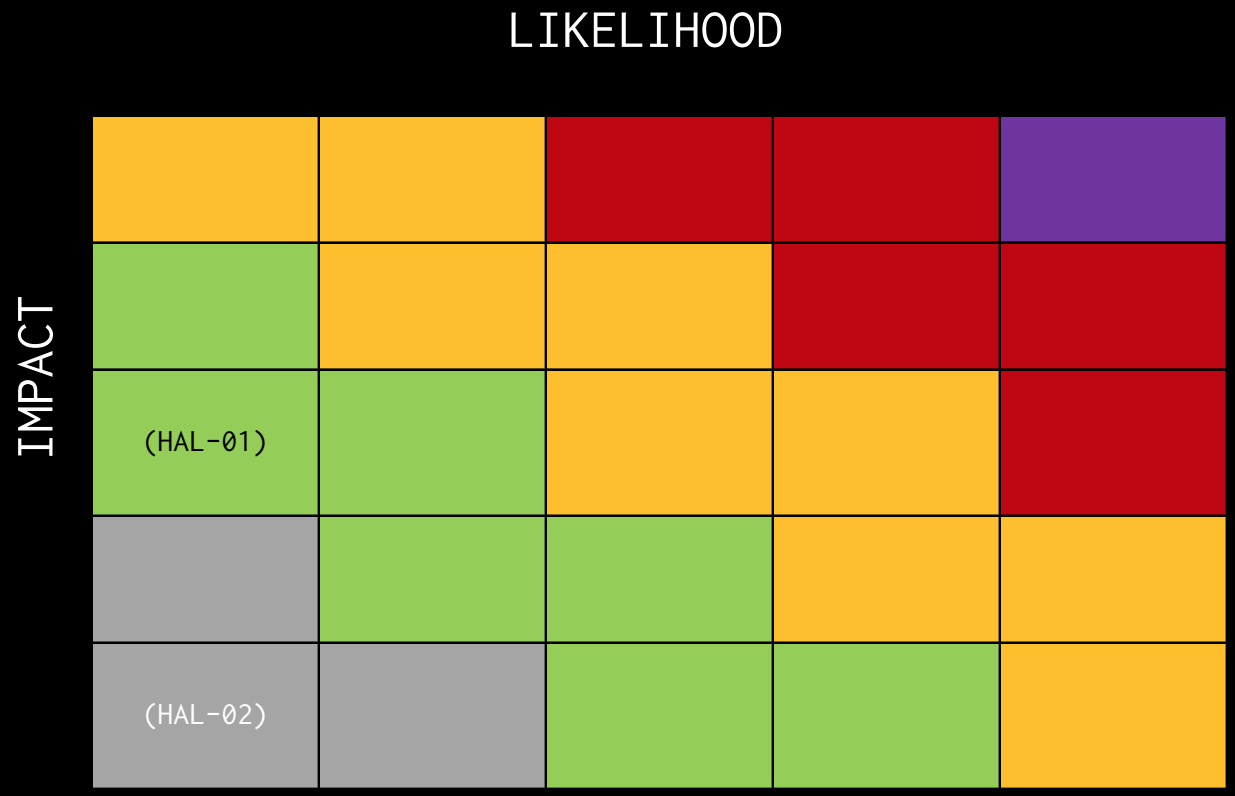
1.4 SCOPE

1. CosmWasm Smart Contracts
   (a) Repository: brotocol-token-contracts
   (b) Git tag:
      i. v1.0.0-audit.6
   (c) Contracts in scope:
      i. bonding-v1
      ii. staking-v1
   (d) Functionalities in scope:
      i. Long Term Bonding

Out-of-scope: Code of contracts related to the commits above, but not related to Long Term Bonding, external libraries and financial related attacks.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	1	1



EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) LACK OF ADDRESS NORMALIZATION	Low	RISK ACCEPTED
(HAL-02) UNMANTAINED DEPENDENCY	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) LACK OF ADDRESS NORMALIZATION - LOW

Description:

The multiple features of the bonding-v1 and staking-v1 contracts do not consider that Terra addresses are valid both upper and all lower case. Although valid, a strict comparison between the same address in its all uppercase version (e.g.: **TERRA1KG. . . XNL8**) and its all lowercase version (e.g.: **terra1kg. . . xnl8**) failure.

The likelihood of this issue was reduced as the affected functions were owner-only functionalities, therefore much less prone to error, or queries. Queries affected by this issue will only cause inconvenience rather than a security issue.

Undesired situations could occur, such as loss of control over the dependent contract addresses in case of instantiation with a misvalidated address made up of capital letters. Since update_config does not provide an option for changing some addresses of the contracts, the administrator will lose access to contract management.

Code Location:

Listing 1: contracts/staking-v1/src/contract.rs (Lines 60,61,62,63)

```
43 pub fn instantiate(
44     deps: DepsMut,
45     env: Env,
46     _info: MessageInfo,
47     msg: InstantiateMsg,
48 ) -> Result<Response, ContractError> {
49     set_contract_version(deps.storage, CONTRACT_NAME,
↳ CONTRACT_VERSION)?;
50
51     let community_bonding_contract = if let Some(addr) = msg.
↳ community_bonding_contract {
52         Some(deps.api.addr_canonicalize(&addr)?)
```

```
53      } else {
54          None
55      };
56
57      let config = Config {
58          owner: deps.api.addr_canonicalize(&msg.owner)?,
59          paused: false,
60          bro_token: deps.api.addr_canonicalize(&msg.bro_token)?,
61          rewards_pool_contract: deps.api.addr_canonicalize(&msg.
↳ rewards_pool_contract)?,
62          bbro_minter_contract: deps.api.addr_canonicalize(&msg.
↳ bbro_minter_contract)?,
63          epoch_manager_contract: deps.api.addr_canonicalize(&msg.
↳ epoch_manager_contract)?,
64          community_bonding_contract,
65          unstake_period_blocks: msg.unstake_period_blocks,
66          min_staking_amount: msg.min_staking_amount,
67          lockup_config: LockupConfig {
68              min_lockup_period_epochs: msg.min_lockup_period_epochs
↳ ,
69              max_lockup_period_epochs: msg.max_lockup_period_epochs
↳ ,
70              base_rate: msg.base_rate,
71              linear_growth: msg.linear_growth,
72              exponential_growth: msg.exponential_growth,
73          },
74      };
```

The update_config function from staking-v1/src/commands.rs does not provide the possibility of changing the above contract addresses. The only address that can be updated is community_bonding_contract:

Listing 2: contracts/staking-v1/src/commands.rs (Line 575)

```
565 pub fn update_config(
566     deps: DepsMut,
567     paused: Option<bool>,
568     unstake_period_blocks: Option<u64>,
569     min_staking_amount: Option<Uint128>,
570     min_lockup_period_epochs: Option<u64>,
571     max_lockup_period_epochs: Option<u64>,
572     base_rate: Option<Decimal>,
573     linear_growth: Option<Decimal>,
```

```
574     exponential_growth: Option<Decimal>,
575     community_bonding_contract: Option<String>,
576 ) -> Result<Response, ContractError>
```

The above-mentioned lack of normalization was also noted in the following lines of the contracts:

Listing 3: Affected resources

```
1 contracts/bonding-v1/src/commands.rs: #346, 354, 362, 370, 445,
↳ 477
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

One of the two approaches detailed below should be used:

- Update the cosmwasm-vm and use cosmwasm_std::Api::addr_validate (reference CWA-2022-002).
- If the update mentioned is not possible, addresses could be stored in canonical format by using the cosmwasm_std::Api::addr_canonicalize utility function.

The following considerations should be considered when implementing the second option:

- To successfully compare a canonical address, both ends should be in canonical format. For example, when performing access controls, the sender (e.g.: info.sender or env.message.sender) should be canonicalized beforehand too.

- To send funds to a canonicalized address or include them into a message to a different contract, they should be first turn into its human-readable format via the cosmwasm_std::Api::addr_humanize utility function

Remediation plan:

RISK ACCEPTED: The Brokkr team decided not to take action on this finding because the affected functionalities were only available to the contract owner.

3.2 (HAL-02) UNMANTAINED DEPENDENCY - INFORMATIONAL

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. 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	biginit is unmaintained, use uint instead

Code Location:

Listing 4: Dependency tree

```
bigint 4.4.3
 cosmwasm-bignumber 2.2.0
     brotocol-staking-v1 1.1.0
     brotocol-oracle 1.0.0
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Beware of using dependencies and packages that are no longer supported by the developers or have publicly known security flaws, even when not exploitable at the moment.

Remediation plan:

ACKNOWLEDGED: The Brokkr team acknowledged this finding.

THANK YOU FOR CHOOSING

// HALBORN