// HALBORN

Astroport.fi Maker Contract

CosmWasm Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **February 9th, 2022 - February 18th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	02/09/2022	Jose C. Ramirez
0.2	Document Updates	02/18/2022	Jose C. Ramirez
0.3	Draft Version	02/21/2022	Jose C. Ramirez
0.4	Draft Review	02/21/2022	Gabi Urrutia
1.0	Remediation Plan	02/28/2022	Jose C. Ramirez
1.1	Remediation Plan Review	02/28/2022	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
Jose C. Ramirez	Halborn	jose.ramirez@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Astroport.fi engaged Halborn to conduct a security audit on their smart contracts beginning on February 9th and ending on February 18th. The security assessment was scoped to the smart contracts provided to the Halborn team.

1.2 AUDIT SUMMARY

The team at Halborn was provided two weeks 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 improvements to reduce the likelihood and impact of risks, which were mostly addressed by Astroport.fi . The main one being the lack of validation during Bridge removal.

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 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

1.4 SCOPE

Code repository: tokenomics-maker

1. CosmWasm Maker Smart Contract
    (a) Commit ID: bd8f8599e1b1867b6e6a005bc4cef209699683e6
    (b) Contract in scope:
        i. Maker contract

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

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	1	3

LIKELIHOOD

IMPACT

	(HAL-01)			
(HAL-02) (HAL-03) (HAL-04)				

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) LACK OF VALIDATION UPON BRIDGE REMOVAL	Low	SOLVED - 02/28/2022
(HAL-02) MISUSE OF HELPER METHODS	Informational	SOLVED - 02/28/2022
(HAL-03) ARITHMETIC OVERFLOW	Informational	SOLVED - 02/28/2022
(HAL-04) OVERFLOW CHECKS NOT SET FOR PROFILE RELEASE	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) LACK OF VALIDATION UPON BRIDGE REMOVAL - LOW

Description:

AssetInfo instances do not normalize capitalization, taking as different assets ULUNA and uluna. When using the update_bridges function to remove bridges, if a different capitalization was used on the elements of the remove parameter the operation resulted in no modifications being done without the sender receiving any error feedback, as those won't be found in the current list of bridges.

In case the owner unwillingly included incorrect assets to remove the bridge, the undesired bridge would still be available to perform token swapping by the contract, potentially affecting the contract's tokenomics.

Code Location:

Listing 1: contracts/tokenomics/maker/src/contract.rs (Line 688)

```
685     // remove old bridges
686     if let Some(remove_bridges) = remove {
687         for asset in remove_bridges {
688             BRIDGES.remove(deps.storage, asset.to_string());
689         }
690     }
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

Validate the assets provided in the `remove` parameter to ensure that all of them follow proper capitalization. In addition, an error could be raised if the asset to be removed if they are not found on the `BRIDGES` state variable.

Remediation plan:

SOLVED: The issue was fixed with the above recommendation in commit `b7fa67c4d2429e61139331717639cd8f50cb1629`.

3.2 (HAL-02) MISUSE OF HELPER METHODS - INFORMATIONAL

Description:

The use of the unwrap function is very useful for testing environments because a value is forcibly demanded to get an error (aka panic!) if the “Option” does not have “Some” value or “Result”. Nevertheless, leaving unwrap functions in production environments is a bad practice because 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:

Listing 2: Affected resources

```
contracts/tokenomics/maker/src/contract.rs:801:            let (
    asset, bridge) = item.unwrap();
contracts/tokenomics/maker/src/contract.rs:802:            (String
    ::from_utf8(asset).unwrap(), bridge.to_string())
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to avoid the use of unwrap or expect functions in production environments as it could cause a panic!, crashing the contract without error messages. Some alternatives are possible, such as propagating the error by putting a "?", using unwrap_or / unwrap_or_else / unwrap_or_default functions, or using error-chain crate for errors

Reference: https://crates.io/crates/error-chain

Remediation plan:

SOLVED: The issue was fixed with the above recommendation in commit b7fa67c4d2429e61139331717639cd8f50cb1629.

3.3 (HAL-03) UNCHECKED MATH - INFORMATIONAL

Description:

In computer programming, an overflow occurs when an arithmetic operation attempts to create a numeric value that is outside of the range that can be represented with a given number of bits -- either larger than the maximum or lower than the minimum representable value.

This issue has been raised as informational only, as it was not possible to define a clear exploitation scenario. However, the affected lines resembled potentially risky patterns and therefore has been highlighted for consideration.

Code Location:

Listing 3: Affected resources

```
1 contracts/tokenomics/maker/src/contract.rs:520:            Uint128
    ::from(blocks_passed) * astro_distribution_portion,
2 contracts/tokenomics/maker/src/contract.rs:525:        amount -=
    remainder_reward;
3 contracts/tokenomics/maker/src/contract.rs:529:        amount +=
    current_preupgrade_distribution;
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

In the “release” mode, Rust does not panic on overflows and overflown values just “wrap” without any explicit feedback to the user. It is recommended then to use vetted safe math libraries for arithmetic operations

consistently throughout the smart contract system. Consider replacing the addition operator with Rust's `checked_add` method, the subtraction operator with Rust's `checked_subs` method, and so on.

Remediation plan:

SOLVED: The issue was fixed with the above recommendation in commit `b7fa67c4d2429e61139331717639cd8f50cb1629`.

3.4 (HAL-04) 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 package, which could lead to unexpected consequences if the project is refactored.

Code Location:

Listing 4: Affected resources

```
1 contracts/tokenomics/maker/Cargo.toml
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

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

Remediation plan:

ACKNOWLEDGED: Astroport acknowledged this finding.

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. 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

THANK YOU FOR CHOOSING

// HALBORN