// HALBORN

Mars Protocol - Periphery Airdrop

CosmWasm Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **September 12th, 2022 - September 16th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	09/12/2022	Connor Taylor
0.2	Draft Version	09/16/2022	Connor Taylor
0.3	Draft Review	09/16/2022	Gabi Urrutia
1.0	Remediation Plan	09/22/2022	Connor Taylor
1.1	Remediation Plan Review	09/22/2022	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
Connor Taylor	Halborn	Connor.Taylor@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Mars Protocol engaged Halborn to conduct a security audit on their smart contracts beginning on September 12th and ending on September 16th. The security assessment was scoped to the smart contracts provided to the Halborn team.

1.2 AUDIT SUMMARY

The team at Halborn was provided one week 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 found the contract to follow secure development best practices, resulting in only informational findings with negligible security impact.

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

1. CosmWasm Smart Contracts
    (a) Smart Contract: airdrop
    (b) Commit ID: cd5914b25072060d4cb423766ea00ac992a59844
    (c) Contracts in scope:
        i. airdrop

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

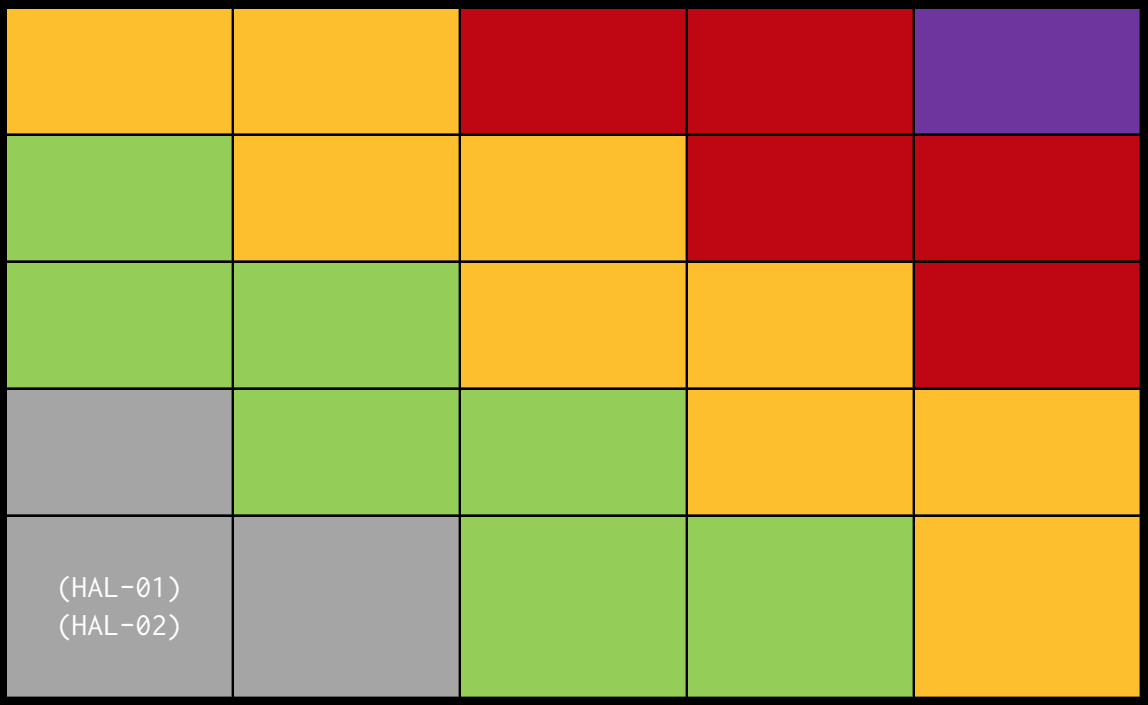
Remediation's Commit ID: b206f007a2ba7eeae5c99ce11df62f3aeebe1030

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	0	2

LIKELIHOOD

IMPACT



EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) OVERFLOW CHECKS NOT SET FOR PROFILE RELEASE	Informational	SOLVED - 09/20/2022
(HAL-02) OUTDATED RUST EDITION IN USE	Informational	SOLVED - 09/20/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) OUTDATED RUST EDITION IN USE - INFORMATIONAL

Description:

A review of the Edition parameter within the contracts **Cargo.toml** file showed that the outdated **2018** Rust version had been included. The Rust team released the latest implementation of the Rust language in October 2021, known as the **2021 edition**. The latest iteration of the language includes many stability and feature improvements which may allow developers to implement more consistent, readable and secure code.

Code Location:

Listing 1: Resources affected

```
1  /contracts/airdrop/Cargo.toml
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to review functionality introduced within the **2021 edition** of the Rust language and, where compatible, upgrade to the latest edition.

Remediation Plan:

SOLVED: The Mars team upgraded the Rust edition to the latest version (2021).

3.2 (HAL-02) OVERFLOW CHECKS NOT SET FOR PROFILE RELEASE - INFORMATIONAL

Description:

Although the overflow-checks parameter is set to **true** in profile.release and implicitly applied to all contracts and packages in the workspace, it is not explicitly enabled in **Cargo.toml** for each individual contract and package, which could have unexpected consequences if the project is refactored.

Code Location:

Listing 2: Resources affected

```
1 contracts/airdrop/Cargo.toml
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended that you explicitly enable overflow checks on each individual contract and package. That measure helps when the project is refactored to avoid unintended consequences.

Remediation Plan:

SOLVED: The overflow-checks parameter has been enforced at both root and module levels.

THANK YOU FOR CHOOSING

// HALBORN