// HALBORN

Mars Protocol CW-Asset

CosmWasm Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **January 13, 2022 - January 21, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	01/13/2022	Jose C. Ramirez
0.2	Document Updates	01/20/2022	Jose C. Ramirez
0.3	Draft Version	01/21/2022	Jose C. Ramirez
0.4	Draft Review	01/21/2022	Gabi Urrutia
1.0	Remediation Plan	02/08/2022	Jose C. Ramirez
1.1	Remediation Plan Review	02/08/2022	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 AUDIT SUMMARY

Mars Protocol engaged Halborn to conduct a security assessment on a CosmWasm smart contract beginning on January 13th, 2022 and ending January 21st, 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 haven't identified relevant risks as common best practices were followed on the development. The only detail worth of mention was the lack of validation on native coins 'denom' field that was properly addressed by Mars Protocol.

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.

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

1.3 SCOPE

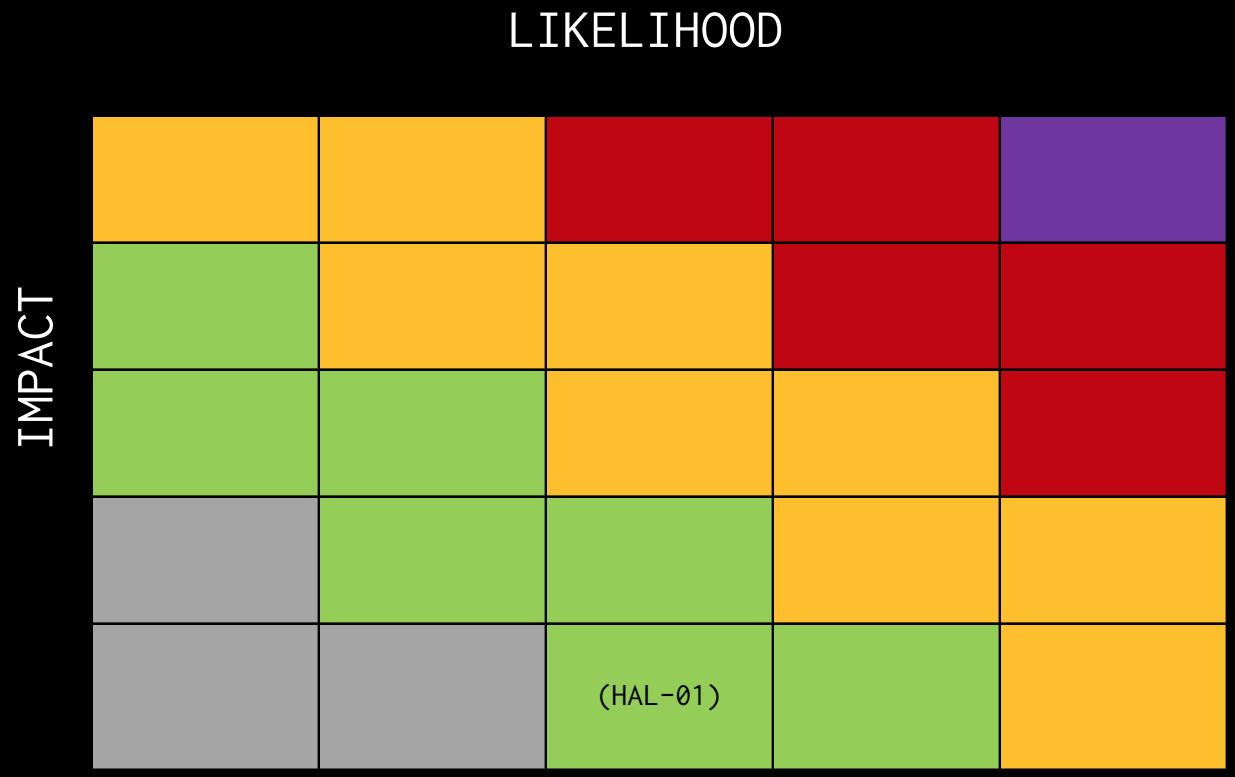
Code repository: https://github.com/mars-protocol/cw-asset

1. CosmWasm Smart Contracts - CW-Asset
    (a) Commit ID: 1795e1698d4a346b9116b1e53be4d10465c506d4
    (b) Files in scope:
        i. asset_info.rs
        ii. asset_list.rs
        iii. asset.rs

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

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	1	0



EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) LACK OF DENOM VALIDATION ON CHECKED ASSET	Low	SOLVED - 02/08/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) LACK OF DENOM VALIDATION ON CHECKED ASSET - LOW

Description:

Assets' information could be stored both as AssetInfoUnchecked and AssetInfo. Validation of an asset's info was done through the check() function part of the AssetInfoUnchecked implementation. However, only concerning CW20 tokens there were actual validation steps being applied but none for Native coins.

Using some format rules upon "checking" an AssetInfoUnchecked element avoids potentially undesirable situations. For instance, if a user creates (mistakenly or not) a fake native coin 'UUSD' instead of 'uusd', it will be stored in contract's storage. As a result, when other users' operations use this fake coin, they will always fail and make users spend transactions fees needlessly.

Listing 1: Proof of concept (Lines 3)

```
1 let uusd = Asset::native("uusd", 5u128);
2 let uusd2 = Asset::native("UUSD", 5u128);
3 assert_eq!(uusd2 == uusd, true); //This will fail as they are
      deemed to be different assets.
```

Code Location:

Listing 2: asset_info.rs (Lines 88)

```
83 pub fn check(&self, api: &dyn Api) -> StdResult<AssetInfo> {
84     Ok(match self {
85         AssetInfoUnchecked::Cw20(contract_addr) => {
86             AssetInfo::Cw20(api.addr_validate(contract_addr)?)
87         }
88         AssetInfoUnchecked::Native(denom) => AssetInfo::Native(
               denom.clone()),
89     })
```

Risk Level:

Likelihood - 3
Impact - 1

Recommendation:

Enforce some basic format rules on denom such as being 4 lowercase a-z characters. A more strict option would be to perform white-listing by comparing the denom against the complete list of valid native coins on Terra.

Remediation plan:

SOLVED: the check function was modified to perform white-listing by accepting a list of valid denoms to compare the asset's details with, as suggested above.

This issue was fixed on commit 9beba1158f8b3e7f06a237c7d35fc89fb1ba3e6b

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.

No security issues were flagged, just the following warning:
- sha2 0.9.8 is yanked

Listing 3: Dependency tree

```
sha2 0.9.8
|- k256 0.9.6
|   |-- cosmwasm-crypto 0.16.2
|       |-- cosmwasm-std 0.16.2
|           |-- terra-cosmwasm 2.2.0
|           |   |-- astroport 0.3.1
|           |       |-- cw-asset 0.3.4
|           |-- cw20 0.9.1
|           |-- cw20 0.8.1
|           |-- cw0 0.9.1
|           |-- cw0 0.8.1
|           |-- cw-storage-plus 0.8.1
|           |   |-- astroport 0.3.1
|           |-- cw-asset 0.3.4
|           |-- astroport 0.3.1
|-- ed25519-zebra 2.2.0
    |-- cosmwasm-crypto 0.16.2
```

THANK YOU FOR CHOOSING

// HALBORN