Mars Protocol
Fields of Mars

CosmWasm Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **January 17th, 2022 - January 28th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	01/17/2022	Michal Bazyli
0.2	Document Updates	01/26/2022	Michal Bazyli
0.3	Draft Version	01/28/2022	Michal Bazyli
0.4	Draft Review	01/31/2022	Gabi Urrutia
1.0	Remediation Plan	02/09/2022	Michal Bazyli
1.1	Remediation Plan Review	02/09/2022	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
Michal Bazyli	Halborn	Michal.Bazyli@halborn.com

EXECUTIVE OVERVIEW

1.1 AUDIT SUMMARY

Mars Protocol engaged Halborn to conduct a security assessment on CosmWasm smart contracts beginning on January 17th, 2022 and ending January 28th, 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 impact of risks, which were mostly addressed by Mars team. The main ones are the following:

- Enforce the use of a valid routine in update_config
- Enforce check of arithmetic 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 regarding 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

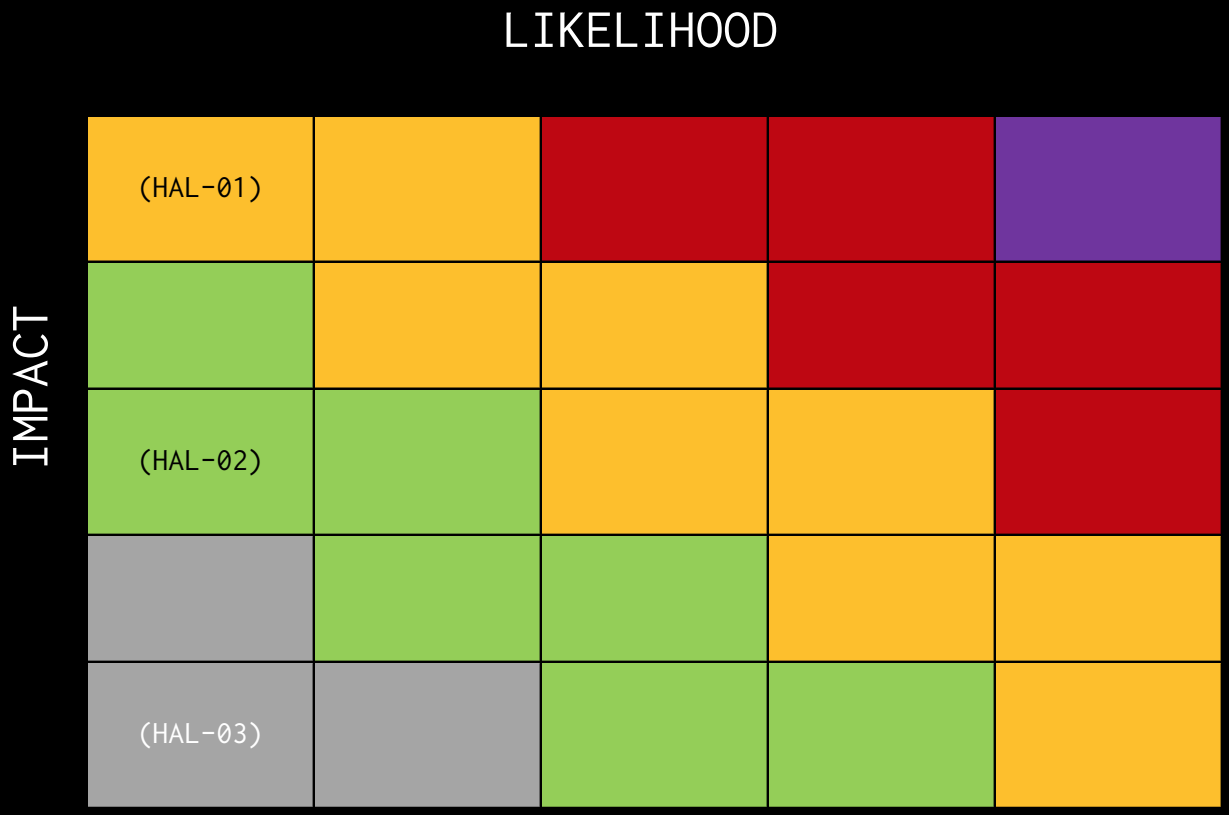
1.3 SCOPE

1. CosmWasm Smart Contracts
   (a) Repository: fields-of-mars
   (b) Commit ID: dc2245ada036d7cfef94a82dd121474d1dd033f2
   (c) Contracts in scope:
      i. martian-field

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

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	1	1



EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) CONFIG PARAMETERS VALUE CAN BE CHANGED UNRESTRICTEDLY	Medium	SOLVED - 02/09/2022
(HAL-02) SOME RATES COULD BE SET TO VALUES GREATER THAN 1	Low	SOLVED - 02/09/2022
(HAL-03) MULTIPLE INSTANCES OF UNCHECKED ARITHMETIC	Informational	SOLVED - 02/09/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) CONFIG PARAMETERS VALUE CAN BE CHANGED UNRESTRICTEDLY - MEDIUM

Description:

`instantiate` and `update_config` functions in **contracts/martian-field/src/execute.rs** allow contract's owner to update `max_ltv`, `bonus_rate` and `fee_rate` fields with a potential unfair amount. This situation can produce the following consequences:

- A malicious (or compromised) owner can change temporarily `max_ltv`to very low rate e.g., : `0.01` and `bonus_rate` to e.g., : `0` and liquidate all positions, draining users assets.
- Owner could mistakenly change `max_ltv` rate to lower than released one, which could become current positions into "unhealthy" ones and ready to be liquidated.
- If `fee_rate` is equal to 1, harvest operations will cause unfair reward distributions and owner could drain user assets. Furthermore, if `fee_rate` is higher than 1 it will cause an overflow.

It is worth noting that likelihood for this to happen is low because **martian-field** contract is intended to be owned by governance (Council) indefinitely, who is the responsible one for this operation.

Code Location:

Listing 1: contracts/martian-field/src/execute.rs (Line 321)

```
314     pub fn update_config(deps: DepsMut, info: MessageInfo,
            new_config: Config) -> StdResult<Response> {
315         let config = CONFIG.load(deps.storage)?;
316
317         if info.sender != config.governance {
318             return Err(StdError::generic_err("only governance can
                    update config"));
```

```
319         }
320
321         CONFIG.save(deps.storage, &new_config)?;
322
323         Ok(Response::default())
324     }
```

Listing 2: contracts/martian-field/src/execute.rs (Lines 248,285)

```
232     pub fn liquidate(
233     deps: DepsMut,
234     env: Env,
235     info: MessageInfo,
236     user_addr: Addr,
237 ) -> StdResult<Response> {
238     let config = CONFIG.load(deps.storage)?;
239     let state = STATE.load(deps.storage)?;
240     let position = POSITION.load(deps.storage, &user_addr).
            unwrap_or_default();
241
242     // position must be active (LTV is not `None`) and the LTV
            must be greater than `max_ltv`
243     let health = compute_health(&deps.querier, &env, &config, &
            state, &position)?;
244
245     // if `health.ltv` is `Some`, it must be greater than `max_ltv
            `
246     // if `health.ltv` is `None`, indicating the position is
            already closed, then it is not liquidatable
247     let ltv = health.ltv.ok_or_else(|| StdError::generic_err("
            position is already closed"))?;
248     if ltv <= config.max_ltv {
249         return Err(StdError::generic_err("position is healthy"));
250     }
251
252     // 1. unbond the user's liquidity tokens from Astro generator
253     // 2. burn liquidity tokens, withdraw primary + secondary
            assets from the pool
254     // 3. swap all primary assets to secondary assets
255     // 4. repay all debts
256     // 5. among all remaining assets, send the amount
            corresponding to `bonus_rate` to the liquidator
257     // 6. refund all assets that're left to the user
258     //
```

```
259     // NOTE: in the previous versions, we sell **all** primary
            assets, which is not optimal because
260     // this will incur bigger slippage, causing worse liquidation
            cascade, and be potentially lucrative
261     // for sandwich attackers
262     //
263     // now, we calculate how much additional secondary asset is
            needed to fully pay off debt, and
264     // reverse-simulate how much primary asset needs to be sold
265     //
266     // TODO: add slippage checks to the swap step so that
            liquidation cannot be sandwich attacked
267     let callbacks = [
268         CallbackMsg::Unbond {
269             user_addr: user_addr.clone(),
270             bond_units_to_reduce: position.bond_units,
271         },
272         CallbackMsg::WithdrawLiquidity {
273             user_addr: user_addr.clone(),
274         },
275         CallbackMsg::Cover {
276             user_addr: user_addr.clone(),
277         },
278         CallbackMsg::Repay {
279             user_addr: user_addr.clone(),
280             repay_amount: health.debt_value,
281         },
282         CallbackMsg::Refund {
283             user_addr: user_addr.clone(),
284             recipient_addr: info.sender.clone(),
285             percentage: config.bonus_rate,
286         },
287         CallbackMsg::Refund {
288             user_addr: user_addr.clone(),
289             recipient_addr: user_addr.clone(),
290             percentage: Decimal::one(),
291         },
292     ];
293
```

Risk Level:

Likelihood - 1
Impact - 5

Recommendation:

Add a validation routine inside instantiate and update_config functions to ensure that:

- Value of fee_rate is lesser than a **maximum threshold** hardcoded in contract.
- Value of max_ltv and bonus_rate is between **minimum and maximum** values hardcoded in the contract.

Remediation plan:

SOLVED: The issue was fixed in commit 0f9c959931fcde3ddf5cdb1907c9177f69284e31.

3.2 (HAL-02) SOME RATES COULD BE SET TO VALUES GREATER THAN 1 - LOW

Description:

The instantiate and update_config functions in **contracts/martian-field/src/execute.rs** do not restrict that rates fields are lesser than 1.

If they are not correctly set, some operations will always panic and won't allow legitimate users to harvest or liquidate; thus generating a denial of service (DoS). The affected fields are the following:

- max_ltv
- fee_rate
- bonus_rate

Code Location:

Listing 3: contracts/martian-field/src/execute.rs (Line 321)

```
314     pub fn update_config(deps: DepsMut, info: MessageInfo,
            new_config: Config) -> StdResult<Response> {
315         let config = CONFIG.load(deps.storage)?;
316
317         if info.sender != config.governance {
318             return Err(StdError::generic_err("only governance can
                    update config"));
319         }
320
321         CONFIG.save(deps.storage, &new_config)?;
322
323         Ok(Response::default())
324     }
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

Add a validation routine inside `instantiate` and `update_config` functions to ensure that **aforementioned fields** are lesser than 1.

Remediation plan:

SOLVED: The issue was fixed in the following commits:

- 2e82ec4798233f14d32a438b5d0238ac1f11583f
- 816db544f50959de79d09cd03f1bfa15e6ef3c86

3.3 (HAL-03) MULTIPLE INSTANCES OF UNCHECKED ARITHMETIC - INFORMATIONAL

Description:

While many instances of checked arithmetic were observed, some calculations omitted these checks. The additional verification performed when using the checked functions ensures that under/overflow states are caught and handled appropriately.

While these instances were not found to be directly exploitable, they should be reviewed to ensure a defence-in-depth approach is achieved.

Code Location:

Listing 4: Resources affected

```
execute_callbacks.rs  (#L237,390, 391,420,478,536)
execute.rs  (#L175)
health.rs (#L51,52,53)
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider using the `checked_add`, `checked_sub` or `checked_mul` methods instead of addition, subtraction, and multiplication operators respectively, in all instances to handle overflows gracefully.

Remediation plan:

SOLVED: Commit ce52053d3a1897b797656a3e60235bdd52147627 fixed the security issue. It is worth noting that there are some arithmetic operations listed above that do not need `checked_*` methods because they are multiplications between `Uint128` and `Decimal`, which invoke `Uint128::multiply_ratio` under the hood:

Listing 5: Resources with no checked_* methods

```
execute_callbacks.rs  (#L390, 391,536)
execute.rs  (#L175)
health.rs (#L51,52,53)
```

THANK YOU FOR CHOOSING

// HALBORN