Mars Protocol Core Contracts

CosmWasm Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **December 2nd, 2021 - February 11th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	12/02/2021	Luis Quispe Gonzales
0.2	Document Updates	12/22/2021	Michal Bazyli
0.3	Document Updates	01/05/2022	Connor Taylor
0.4	Document Updates	01/10/2022	Jose C. Ramirez
0.5	Document Updates	01/21/2022	Luis Quispe Gonzales
0.6	Draft Version	02/11/2022	Luis Quispe Gonzales
0.7	Draft Review	02/15/2022	Gabi Urrutia
1.0	Remediation Plan	02/17/2022	Luis Quispe Gonzales
1.1	Remediation Plan Review	02/18/2022	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
Luis Quispe Gonzales	Halborn	Luis.QuispeGonzales@halborn.com

EXECUTIVE OVERVIEW

1.1 AUDIT SUMMARY

Mars Protocol engaged Halborn to conduct a security assessment on CosmWasm smart contracts beginning on December 2nd, 2021 and ending February 11th, 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:

- Remove AstroportSpot and AstroportLiquidityToken as price sources.
- Handle correctly the cases where debt and collateral are the same asset for liquidations.
- Validate if a slash event with the same key has already been created before.
- Update the logic of some functions to turn recipient address into lower case, in order to avoid tokens to get locked.
- Fix the calculation of total_mars_for_claimers when transferring Mars tokens.

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

1. CosmWasm Smart Contracts
   (a) Repository: mars-core
   (b) Commit ID: fcaa6ffe918a0890a1a6c57f26edb6f8feb25633
   (c) Contracts in scope:
      i. mars-address-provider
      ii. mars-council
      iii. mars-incentives
      iv. mars-ma-token
      v. mars-oracle
      vi. mars-protocol-rewards-collector
      vii. mars-red-bank
      viii. mars-safety-fund
      ix. mars-staking
      x. mars-treasury
      xi. mars-vesting
      xii. mars-xmars-token

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

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	6	5	10	7

LIKELIHOOD

IMPACT

(HAL-08)		(HAL-03)	(HAL-01)	
(HAL-12) (HAL-13) (HAL-14)	(HAL-09) (HAL-10)		(HAL-04) (HAL-05) (HAL-06)	(HAL-02)
(HAL-18) (HAL-19) (HAL-20)	(HAL-15)	(HAL-11)	(HAL-07)	
(HAL-22) (HAL-23) (HAL-24)	(HAL-21)	(HAL-16) (HAL-17)		
(HAL-25) (HAL-26) (HAL-27) (HAL-28)				

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) SOME PRICE SOURCES DO NOT PREVENT MANIPULATION OF ASSETS PRICE IN THE ORACLE	High	RISK ACCEPTED
(HAL-02) LIQUIDATION CAN TOTALLY CORRUPT THE VALUES OF TOTAL DEBT, INDEXES AND RATES	High	SOLVED - 02/02/2022
(HAL-03) SLASH EVENTS CAN BE OVERWRITTEN WHEN TRANSFERRING MARS TOKENS	High	SOLVED - 02/17/2022
(HAL-04) MARS TOKENS CAN GET LOCKED IN CONTRACT WHEN UNSTAKING	High	SOLVED - 02/11/2022
(HAL-05) TOTAL MARS FOR CLAIMERS IS MISCALCULATED WHEN TRANSFERRING MARS TOKENS	High	SOLVED - 02/09/2022
(HAL-06) TOKENS GET LOCKED WHEN TRANSFERRING TO UPPER-CASE ADDRESSES	High	FUTURE RELEASE
(HAL-07) POSSIBILITY TO LIQUIDATE WHEN COLLATERAL ASSET IS UNSET	Medium	SOLVED - 01/10/2022
(HAL-08) NO MINIMUM THRESHOLD FOR SOME PARAMETERS OF COUNCIL CONFIGURATION	Medium	PARTIALLY SOLVED
(HAL-09) LOAN LIMIT CAN BE UPDATED FOR USERS WITH COLLATERALIZED DEBTS	Medium	SOLVED - 01/12/2022
(HAL-10) RESTRICTION TO NOT SWAP MARS TOKENS CAN BE BYPASSED	Medium	SOLVED - 02/10/2022
(HAL-11) POSSIBILITY TO DEPOSIT, REPAY OR LIQUIDATE WITH NATIVE COINS NOT REGISTERED IN STORAGE	Medium	SOLVED - 01/10/2022
(HAL-12) FUNCTIONS WITH EXCESSIVE PRIVILEGES	Low	RISK ACCEPTED
(HAL-13) MISSING CHECK FOR ASSETS WITH ASTROPORTLIQUIDITYTOKEN AS PRICE SOURCE	Low	RISK ACCEPTED
(HAL-14) PRIVILEGED ADDRESS CAN BE TRANSFERRED WITHOUT CONFIRMATION	Low	RISK ACCEPTED

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-15) USERS CAN BURN THEIR OWN XMARS TOKENS	Low	RISK ACCEPTED
(HAL-16) QUERYING PRICE COULD PANIC FOR ASSETS WITH TWAP PRICE SOURCES	Low	RISK ACCEPTED
(HAL-17) ROUNDING ISSUES WHEN DISTRIBUTING REWARDS TO PROTOCOL CONTRACTS	Low	RISK ACCEPTED
(HAL-18) LIQUIDATION THRESHOLD VALUE CAN BE CHANGED UNRESTRICTEDLY	Low	RISK ACCEPTED
(HAL-19) REPEATED ASSETS CAN BE INITIALIZED	Low	SOLVED - 01/31/2022
(HAL-20) SOME QUERY MESSAGES DO NOT HANDLE ERRORS ADEQUATELY	Low	RISK ACCEPTED
(HAL-21) FUNCTION TO SWAP ASSET TO UUSD IS NOT RESTRICTED ENOUGH	Low	FUTURE RELEASE
(HAL-22) POTENTIAL ASSET PRICE OVERWRITING	Informational	ACKNOWLEDGED
(HAL-23) OPTIMAL UTILIZATION RATE COULD SKIP VALIDATION OF AN INADEQUATE VALUE	Informational	ACKNOWLEDGED
(HAL-24) UPDATED BORROW INDEX COULD BE WRONGLY CALCULATED	Informational	ACKNOWLEDGED
(HAL-25) INACCURATE ERROR MESSAGES	Informational	SOLVED - 01/10/2022
(HAL-26) MISUSE OF HELPER METHODS	Informational	ACKNOWLEDGED
(HAL-27) OVERFLOW CHECKS NOT SET FOR PROFILE RELEASE	Informational	SOLVED - 01/10/2022
(HAL-28) MULTIPLE INSTANCES OF UNCHECKED ARITHMETIC	Informational	PARTIALLY SOLVED

FINDINGS & TECH DETAILS

3.1 (HAL-01) SOME PRICE SOURCES DO NOT PREVENT MANIPULATION OF ASSETS PRICE IN THE ORACLE - HIGH

Description:

If an asset is created with execute_set_asset function in **contracts/mars-oracle/src/contract.rs** using **AstroportSpot** or **AstroportLiquidityToken** as price source, an attacker can create an imbalance in its corresponding pool in **Astroport AMM protocol** and obtain considerable profits at expenses of Mars markets.

Below is described an attack scenario for an asset created with **AstroportSpot** as price source. Initial balance for the pool is 8M tokens and 2M UST.

Attack scenario:

1. Attacker obtains 3M UST (flash loan or whale) and swap them in **Astroport AMM protocol** to imbalance the **Token-UST** pool.
2. Attacker obtains 4.8M tokens in return for the swap and deposits them as collateral in **mars-red-bank** contract.
3. Attacker can borrow 5.6M UST from **mars-red-bank** contract.
4. Attacker returns flash loan and obtains 2.6M UST as profit at expenses of Mars market.

A proof of concept video showing how to exploit this security issue is included in the report.

Code Location:

If an asset is created with AstroportSpot as price source, its price is queried with **AstroportQueryMsg::Simulation** message, using a potentially imbalanced Astroport pool as information source.

Listing 1: contracts/mars-oracle/src/contract.rs (Line 240)

```
239 PriceSourceChecked::AstroportSpot { pair_address } => {
240     query_astroport_spot_price(&deps.querier, &pair_address)
241 }
```

Listing 2: contracts/mars-oracle/src/contract.rs (Line 403)

```
400 let response: SimulationResponse =
401     querier.query(&QueryRequest::Wasm(WasmQuery::Smart {
402         contract_addr: pair_address.to_string(),
403         msg: to_binary(&AstroportQueryMsg::Simulation {
                offer_asset })?,
404     }))?;
405
406 Ok(Decimal::from_ratio(
407     response.return_amount + response.commission_amount,
408     PROBE_AMOUNT,
409 ))
```

If an asset is created with AstroportLiquidityToken as price source, its price is queried using a potentially imbalanced Astroport pool as information source.

Listing 3: contracts/mars-oracle/src/contract.rs (Lines 291,295,298)

```
287 PriceSourceChecked::AstroportLiquidityToken { pair_address } => {
288     let pool = query_astroport_pool(&deps.querier, &pair_address)?;
289
290     let asset0: Asset = (&pool.assets[0].info).into();
291     let asset0_price = query_asset_price(deps, env.clone(), asset0.
            get_reference())?;
292     let asset0_value = asset0_price * pool.assets[0].amount;
293
294     let asset1: Asset = (&pool.assets[1].info).into();
295     let asset1_price = query_asset_price(deps, env, asset1.
```

```
            get_reference())?;
296     let asset1_value = asset1_price * pool.assets[1].amount;
297
298     let price = Decimal::from_ratio(asset0_value + asset1_value,
            pool.total_share);
299     Ok(price)
300 }
```

Risk Level:

Likelihood - 4
Impact - 5

Recommendation:

Remove **AstroportSpot** and **AstroportLiquidityToken** as price sources for execute_set_asset and query_asset_price functions. As long as those price sources exist in the code (even if not frequently used), they are potential doors for extremely harmful attacks, i.e.: draining funds from Mars markets.

Remediation plan:

RISK ACCEPTED: The Mars team accepted the risk for this finding. They also stated that the mentioned price sources are useful when testing currently and will be comprehensive in the context of the risk that those price sources are not recommended for production. Furthermore, they will consider removing it entirely in future versions of the protocol if the community agrees.

3.2 (HAL-02) LIQUIDATION CAN TOTALLY CORRUPT THE VALUES OF TOTAL DEBT, INDEXES AND RATES - HIGH

Description:

execute_liquidate function in **contracts/mars-red-bank/src/contract.rs** does not handle adequately the cases where **debt** and **collateral** are the same asset because changes on debt_market are overridden by changes on collateral_market. The values that can be totally corrupted on debt_market are the following:

- debt_total_scaled
- borrow_index
- liquidity_index
- indexes_last_updated
- borrow_rate
- liquidity_rate

Proof of concept:

1. User liquidates using the same asset for **collateral** and **debt**:

```
let liquidate_msg: ExecuteMsg = ExecuteMsg::Receive(Cw20ReceiveMsg {
    msg: to_binary(data: &ReceiveMsg::LiquidateCw20 {
        /*collateral_asset: Asset::Native {
            denom: "collateral".to_string(),
        },*/
        collateral_asset: Asset::Cw20 {
            contract_addr: cw20_debt_contract_addr.clone().to_string(),
        },
        user_address: user_address.to_string(),
        receive_ma_token: true,
    }): Result<Binary, StdError>
    .unwrap(),
    sender: liquidator_address.to_string(),
    amount: first_debt_to_repay.into() ,
});
```

2. The value of debt_total_scaled is updated with the amount repaid:

```
1401        debt_market.debt_total_scaled = debt_market: Market
1402            .debt_total_scaled: Uint128
1403            .checked_sub(debt_amount_to_repay_scaled)?;
1404
```

3. Changes on debt_market are overridden by changes on collateral_market because **debt_asset_reference** and **collateral_asset_reference** have the same value:

```
// save markets
MARKETS.save(store: deps.storage, k: debt_asset_reference.as_slice(), data: &debt_market)?;
MARKETS.save(
    store: deps.storage,
    k: collateral_asset_reference.as_slice(),
    data: &collateral_market,
)?;
```

4. When testing, it was identified that value of debt_total_scaled had not been updated correctly:

```
2587            assert_eq!(
2588                expected_global_cw20_debt_scaled,
2589                debt_market_after.debt_total_scaled
2590            );
```

```
running 1 test
thread 'contract::tests::hacking_execute_liquidate' panicked at 'assertion failed: `(left == right)`
  left: `Uint128(1799739104536880)`,
 right: `Uint128(1800000000000000)`', contracts/mars-red-bank/src/contract.rs:2587:13
```

Code Location:

execute_liquidate function receive collateral_asset and deb_asset as arguments of **Asset** type.

Listing 4: contracts/mars-red-bank/src/contract.rs (Lines 1204,1205)

```
1199 pub fn execute_liquidate(
1200     mut deps: DepsMut,
1201     env: Env,
1202     _info: MessageInfo,
1203     liquidator_address: Addr,
1204     collateral_asset: Asset,
1205     debt_asset: Asset,
```

```
206     user_address: Addr,
207     sent_debt_asset_amount: Uint128,
208     receive_ma_token: bool,
209 ) -> Result<Response, ContractError> {
210     let block_time = env.block.time.seconds();
211     let (debt_asset_label, debt_asset_reference, debt_asset_type)
            = debt_asset.get_attributes();
```

Changes on debt_market are overridden in contract's storage by changes on collateral_market.

Listing 5: contracts/mars-red-bank/src/contract.rs (Lines 1416-1419)

```
1414 // save markets
1415 MARKETS.save(deps.storage, debt_asset_reference.as_slice(), &
       debt_market)?;
1416 MARKETS.save(
1417     deps.storage,
1418     collateral_asset_reference.as_slice(),
1419     &collateral_market,
```

Risk Level:

Likelihood - 5
Impact - 4

Recommendation:

Update the logic of execute_liquidate function to handle correctly the cases where **debt** and **collateral** are the same asset.

Remediation plan:

SOLVED: The issue was fixed in the following commits:

- 82c6649db9536fffa67754c151d24eb97b109bc2
- 20d493c4b6b68586dedf43aa0de69989bc587b62

3.3 (HAL-03) SLASH EVENTS CAN BE OVERWRITTEN WHEN TRANSFERRING MARS TOKENS - HIGH

Description:

execute_transfer_mars function in **contracts/mars-staking/src/contract.rs** allows owner to transfer Mars tokens to some recipient. Every time this operation is done, a slash event is created with env.block.height as key and slash_percentage as value.

When users claim, slash events are applied in chronological order to adjust and reduce claim amount to the real value. However if execute_transfer_mars function is called two or more times during the same block, previous slash events will be overwritten by the new ones. As a consequence, users will be able to claim more at expenses of the **mars-staking** contract funds.

Code Location:

Listing 6: contracts/mars-staking/src/contract.rs (Lines 381-382)

```
377 let slash_percentage = Decimal::from_ratio(amount,
      total_mars_in_staking_contract);
378
379 SLASH_EVENTS.save(
380     deps.storage,
381     U64Key::new(env.block.height),
382     &     { slash_percentage },
383 )?;
```

Risk Level:

Likelihood - 3
Impact - 5

Recommendation:

Update the logic of `execute_transfer_mars` function to throw error messages if a slash event with the same key (current `env.block.height`) has already been created before.

Remediation plan:

SOLVED: The issue was fixed in commit `7b5fbcc9301b3cbb195098a2a6edb143abb166fc`.

3.4 (HAL-04) MARS TOKENS CAN GET LOCKED IN CONTRACT WHEN UNSTAKING - HIGH

Description:

When a user calls execute_unstake function in **contracts/mars-staking/src/contract.rs** with a recipient address in upper case (e.g.: **TERRA1KG...XNL8**), the claim is stored in **CLAIMS** with the upper case address as a key.

As a consequence, when the recipient tries to claim with execute_claim function, he won't be able to do it because the claim is loaded from **CLAIMS** using info.sender as a key, which is always in lower case (e.g.: **terra1kg...xnl8**), i.e.: his Mars tokens get locked in contract forever.

Code Location:

If recipient address is in upper case, execute_unstake function will store the claim in **CLAIMS** with the upper case address as a key.

Listing 7: contracts/mars-staking/src/contract.rs (Lines 268,273)

```
267     let recipient = option_recipient.unwrap_or_else(|| staker.clone
            ());
268     let recipient_addr = deps.api.addr_validate(&recipient)?;
269
270     if CLAIMS.may_load(deps.storage, &recipient_addr)?.is_some() {
271         return Err(ContractError::UnstakeActiveClaim {});
272     }
273     CLAIMS.save(deps.storage, &recipient_addr, &claim)?;
```

execute_claim function loads claim from **CLAIMS** using info.sender as a key, which is always in lower case.

Listing 8: contracts/mars-staking/src/contract.rs (Line 307)

```
301 pub fn execute_claim(
302     deps: DepsMut,
303     env: Env,
304     info: MessageInfo,
305     option_recipient: Option<String>,
306 ) -> Result<Response, ContractError> {
307     let mut claim = CLAIMS.load(deps.storage, &info.sender)?;
```

Risk Level:

Likelihood - 4
Impact - 4

Recommendation:

Update the logic of execute_unstake function to turn address into lower case for recipient.

Remediation plan:

SOLVED: The issue was fixed in commit 14fdcc2c207f6528e31952f33eb36c32a4a00bf4.

3.5 (HAL-05) TOTAL MARS FOR CLAIMERS IS MISCALCULATED WHEN TRANSFERRING MARS TOKENS - HIGH

Description:

execute_transfer_mars function in **contracts/mars-staking/src/contract.rs** allows owner to transfer Mars tokens to some recipient. Every time this operation is done, total_mars_for_claimers is updated, but its value is wrongly calculated.

As a consequence, in some scenarios legitimate users will not be able to claim from **mars-staking** contract as expected and operation will panic because of underflow.

Code Location:

Listing 9: contracts/mars-staking/src/contract.rs (Line 386)

```
377 let slash_percentage = Decimal::from_ratio(amount,
        total_mars_in_staking_contract);
378
379 SLASH_EVENTS.save(
380     deps.storage,
381     U64Key::new(env.block.height),
382     &SlashEvent { slash_percentage },
383 )?;
384
385 let mut global_state = GLOBAL_STATE.load(deps.storage)?;
386 global_state.total_mars_for_claimers = global_state.
        total_mars_for_claimers * slash_percentage;
387 GLOBAL_STATE.save(deps.storage, &global_state)?;
```

Risk Level:

Likelihood - 4
Impact - 4

Recommendation:

Fix the logic of `execute_transfer_mars` function to multiply the value of current `total_mars_for_claimers` by `1 - slash_percentage` when trying to update its value.

Remediation plan:

SOLVED: The issue was fixed in commit `956c935553d02bbc7f2a3d0198f312d15e75420f`.

3.6 (HAL-06) TOKENS GET LOCKED WHEN TRANSFERRING TO UPPER-CASE ADDRESSES - HIGH

Description:

When accounts transfer / send **mars-ma-token** or **mars-xmars-token** to a recipient address in upper case (e.g.: **TERRA1KG...XNL8**), the new balance is stored in **BALANCES** with the upper case address as a key.

As a consequence, when the recipient tries to use his tokens, he won't be able to do it because the balance is loaded from **BALANCES** using info.sender as a key, which is always in lower case (e.g.: **terra1kg...xnl8**), i.e.: his tokens get locked forever.

The affected smart contracts are the following:

- mars-ma-token
- mars-xmars-token
- mars-red-bank
- mars-staking

Code Location:

Listing 10: Resources affected

```
mars-ma-token: execute_transfer (recipient_unchecked)
mars-ma-token: execute_transfer_on_liquidation (
    recipient_unchecked)
mars-ma-token: execute_send (contract_unchecked)
mars-ma-token: execute_transfer_from (recipient)
mars-ma-token: execute_send_from (contract)
mars-xmars-token: execute_transfer (recipient)
mars-xmars-token: execute_send (contract)
mars-xmars-token: execute_transfer_from (recipient)
mars-xmars-token: execute_send_from (contract)
mars-red-bank: execute_liquidate (liquidator_address)
```

```
mars-staking: execute_claim (option_recipient)
mars-staking: execute_transfer_mars (recipient_unchecked)
```

Risk Level:

Likelihood - 4
Impact - 4

Recommendation:

Update the logic of functions mentioned above to turn recipient addresses into lower case.

Remediation plan:

PENDING The Mars team stated that in the short term they would analyze how to address this issue comprehensively on the protocol.

3.7 (HAL-07) POSSIBILITY TO LIQUIDATE WHEN COLLATERAL ASSET IS UNSET - MEDIUM

Description:

`execute_liquidate` function in **contracts/mars-red-bank/src/contract.rs** does not restrict that liquidations are done using only assets set as collaterals. As a consequence, users are able to liquidate debts using even collaterals that borrowers have explicitly unset, which could affects borrowers' lending strategy.

Code Location:

`execute_liquidate` function does not verify if `collateral_asset` is set as collateral.

Listing 11: contracts/mars-red-bank/src/contract.rs

```
232 let (collateral_asset_label, collateral_asset_reference,
        collateral_asset_type) =
233     collateral_asset.get_attributes();
234
235 let mut collateral_market =
236     MARKETS.load(deps.storage, collateral_asset_reference.as_slice
            ())?;
237
238 if !collateral_market.active {
239     return Err(ContractError::MarketNotActive {
240         asset: collateral_asset_label,
241     });
242 }
243
244 // check if user has available collateral in specified collateral
        asset to be liquidated
245 let user_collateral_balance_scaled = cw20_get_balance(
246     &deps.querier,
247     collateral_market.ma_token_address.clone(),
248     user_address.clone(),
```

```
249 )?;
250 let user_collateral_balance = get_underlying_liquidity_amount(
251     user_collateral_balance_scaled,
252     &collateral_market,
253     block_time,
254 )?;
255 if user_collateral_balance.is_zero() {
256     return Err(ContractError::
            CannotLiquidateWhenNoCollateralBalance {});
257 }
```

Risk Level:

Likelihood - 4
Impact - 3

Recommendation:

Update the logic of execute_liquidate function to verify if collateral_asset parameter is set as collateral. Otherwise, it should throw an error message.

Remediation plan:

SOLVED: The issue was fixed in commit 256559bb553cbffe6683a596f03a2d4eb2bd0a95.

3.8 (HAL-08) NO MINIMUM THRESHOLD FOR SOME PARAMETERS OF COUNCIL CONFIGURATION - MEDIUM

Description:

validate function from **packages/mars-core/src/council.rs** does not validate that proposal_required_threshold, proposal_effective_delay or proposal_expiration_period from **Config** has a minimum threshold, which could generate the following situations:

- Proposals can be passed without reaching the majority of votes.
- Malicious changes proposed through voting could even be executed immediately, not allowing legitimate users to react timely.
- Proposals could never be executed if expiration period is not appropriately set.

Code Location:

Listing 12: packages/mars-core/src/council.rs (Lines 32,42)

```
31 pub fn validate(&self) -> Result<(), MarsError> {
32   let conditions_and_names = vec![
33       (
34        Self::less_or_equal_one(&self.proposal_required_quorum),
35        "proposal_required_quorum",
36       ),
37       (
38        Self::less_or_equal_one(&self.proposal_required_threshold),
39        "proposal_required_threshold",
40       ),
41   ];
42   all_conditions_valid(conditions_and_names)
43 }
```

Risk Level:

Likelihood - 1
Impact - 5

Recommendation:

Update the logic of validate function to ensure the following conditions:

1. proposal_required_threshold is greater or equal than 0.5.
2. proposal_expiration_period exceeds 0.
3. proposal_effective_delay is greater or equal than a **minimum threshold** that allows Mars users to act timely against any issue that the protocol could have when changes are made. The following are some examples of timelocks used on other protocols:
    - Uniswap: 48-hours timelock
    - Compound: 48-hours timelock

Remediation plan:

PARTIALLY SOLVED: The issue for proposal_required_threshold was fixed in the following commits:

- 4c64bd62088be852d55b94a956884d0ab185c421
- 1f78cc5d4f98cb37548ca6ca1d63029f231ae404

The Mars team also claimed that they will consider adding the remaining restrictions in future versions of the protocol.

3.9 (HAL-09) LOAN LIMIT CAN BE UPDATED FOR USERS WITH COLLATERALIZED DEBTS - MEDIUM

Description:

`execute_update_uncollateralized_loan_limit` function in **contracts/mars-red-bank/src/contract.rs** allows updating the loan limit for users with collateralized debts. As a consequence, their debts would be reset (`amount_scaled = 0`) without have been repaid or liquidated by someone else, which greatly affects contract's funds.

It is worth noting that likelihood for this to happen is limited because **mars-red-bank** contract is intended to be owned by governance indefinitely, who is the responsible one for this operation.

Code Location:

Listing 13: contracts/mars-red-bank/src/contract.rs (Lines 636-639)

```
626 UNCOLLATERALIZED_LOAN_LIMITS.save(
627     deps.storage,
628     (asset_reference.as_slice(), &user_address),
629     &new_limit,
630 )?;
631
632 DEBTS.update(
633     deps.storage,
634     (asset_reference.as_slice(), &user_address),
635     |debt_opt: Option<Debt>| -> StdResult<_> {
636         let mut debt = debt_opt.unwrap_or(Debt {
637             amount_scaled: Uint128::zero(),
638             uncollateralized: false,
639         });
640         // if limit == 0 then uncollateralized = false, otherwise
               uncollateralized = true
641         debt.uncollateralized = !new_limit.is_zero();
642         Ok(debt)
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendation:

Update the logic of `execute_update_uncollateralized_loan_limit` function to restrict that loan limit can be updated only for users with no collateralized debts. Otherwise, it should throw an error message.

Remediation plan:

SOLVED: The issue was fixed in commit bc2936b3953ef0006e0511164170ec8aff98746a.

3.10 (HAL-10) RESTRICTION TO NOT SWAP MARS TOKENS CAN BE BYPASSED - MEDIUM

Description:

execute_swap_asset_to_uusd function in **contracts/mars-staking/src/contract.rs** allows users to swap assets apart from Mars token to **UST**. However, this restriction can be bypassed by sending an **AssetInfo** with contract_addr = **address of Mars tokens in upper-case**.

As a consequence, an attacker could swap all (or almost all) Mars tokens to UST and legitimate users won't be able to unstake their xMars tokens or will receive, in return, much less than expected.

Code Location:

Listing 14: contracts/mars-staking/src/contract.rs (Lines 413,426-428)

```
410 pub fn execute_swap_asset_to_uusd(
411     deps: DepsMut,
412     env: Env,
413     offer_asset_info: AssetInfo,
414     amount: Option<Uint128>,
415 ) -> Result<Response, ContractError> {
416     let config = CONFIG.load(deps.storage)?;
417
418     // throw error if the user tries to swap Mars
419     let mars_token_address = address_provider::helpers::
        query_address(
420         &deps.querier,
421         config.address_provider_address,
422         MarsContract::MarsToken,
423     )?;
424
425     if let AssetInfo::Token { contract_addr } = offer_asset_info.
        clone() {
426         if contract_addr == mars_token_address {
427             return Err(ContractError::MarsCannotSwap {});
```

```
428             }
429         }
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendation:

Update the logic of `execute_swap_asset_to_uusd` function to turn addresses in `offer_asset_info` and `mars_token_address` into lowercase before comparing them.

Remediation plan:

SOLVED: The issue was fixed in commit `1c559d34da76bbcacedb18d832280c16037121e4`.

3.11 (HAL-11) POSSIBILITY TO DEPOSIT, REPAY OR LIQUIDATE WITH NATIVE COINS NOT REGISTERED IN STORAGE - MEDIUM

Description:

`get_denom_amount_from_coins` function in **contracts/mars-red-bank/src/contract.rs** accepts native coins apart from the ones registered in markets' storage.

As a consequence, if users mistakenly deposit, repay or liquidate with additional native coins, those coins will keep locked in **mars-red-bank** contract without possibility to withdraw them, instead of throwing error messages for failed operation.

Code Location:

Listing 15: contracts/mars-red-bank/src/contract.rs

```
2114  // native coins
2115  fn get_denom_amount_from_coins(coins: &[Coin], denom: &str) ->
          Uint128 {
2116      coins
2117          .iter()
2118          .find(|c| c.denom == denom)
2119          .map(|c| c.amount)
2120          .unwrap_or_else(Uint128::zero)
2121  }
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

Update the logic of `get_denom_amount_from_coins` function to restrict that `coins` parameter could only have 1 element. Otherwise, it should throw an error message.

Remediation plan:

SOLVED: The issue was fixed in commit `b22e910fcb28d7450b0abc222a719ae9dcd85cd0`.

3.12 (HAL-12) FUNCTIONS WITH EXCESSIVE PRIVILEGES - LOW

Description:

execute_execute_cosmos_msg function in **contracts/mars-incentives/src/contract.rs** and **contracts/mars-protocol-rewards-collector/src/contract.rs** allows owner of those contracts to execute arbitrary messages, which could include transferring all tokens out of contracts to a potential malicious external account.

It is worth noting that likelihood for this to happen is low because **mars-incentives** and **mars-protocol-rewards-collector** contracts are intended to be owned by governance indefinitely, who is the responsible one for this operation.

Code Location:

Listing 16: contracts/mars-incentives/src/contract.rs

```
295 pub fn execute_execute_cosmos_msg(
296     deps: DepsMut,
297     _env: Env,
298     info: MessageInfo,
299     msg: CosmosMsg,
300 ) -> Result<Response, MarsError> {
301     let config = CONFIG.load(deps.storage)?;
302
303     if info.sender != config.owner {
304         return Err(MarsError::Unauthorized {});
305     }
306
307     let response = Response::new()
308         .add_attribute("action", "execute_cosmos_msg")
309         .add_message(msg);
310
311     Ok(response)
312 }
```

Listing 17: contracts/mars-protocol-rewards-collector/src/contract.rs

```
347 pub fn execute_execute_cosmos_msg(
348     deps: DepsMut,
349     _env: Env,
350     info: MessageInfo,
351     msg: CosmosMsg,
352 ) -> Result<Response, MarsError> {
353     let config = CONFIG.load(deps.storage)?;
354
355     if info.sender != config.owner {
356         return Err(MarsError::Unauthorized {});
357     }
358
359     let response = Response::new()
360         .add_attribute("action", "execute_cosmos_msg")
361         .add_message(msg);
362
363     Ok(response)
364 }
```

Risk Level:

Likelihood - 1
Impact - 4

Recommendation:

If not used, it is recommended to remove execute_execute_cosmos_msg function from **mars-incentives** and **mars-protocol-rewards-collector** contracts.

Remediation plan:

RISK ACCEPTED: The Mars team accepted the risk of this finding. They also stated that require governance to manipulate the funds that are needed. For example, if the incentives are no longer used, they can take the Mars tokens out of the contract.

3.13 (HAL-13) MISSING CHECK FOR ASSETS WITH ASTROPORTLIQUIDITYTOKEN AS PRICE SOURCE - LOW

Description:

`execute_set_asset` function in **contracts/mars-oracle/src/contract.rs** does not validate if `contract_addr` in **Asset** matches the **LP token** from `pair_address` in assets with **AstroportLiquidityToken** as price source. As a consequence, owner of **mars-oracle** contract could mistakenly set an asset with an erroneous price source.

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

Code Location:

Listing 18: contracts/mars-oracle/src/contract.rs (Line 97)

```
92 match &price_source {
93     PriceSourceChecked::AstroportSpot { pair_address }
94     | PriceSourceChecked::AstroportTwap { pair_address, .. } => {
95         assert_astroport_pool_assets(&deps.querier, &asset,
               pair_address)?;
96     }
97     _ => (),
98 }
```

Risk Level:

Likelihood - 1
Impact - 4

Recommendation:

Update the logic of execute_set_asset function to validate if contract_addr in **Asset** matches the **LP token** from pair_address in assets with **AstroportLiquidityToken**. To fulfill this validation, **QueryMsg::Pair** query message in Astroport pair contract can be used.

Remediation plan:

RISK ACCEPTED: The Mars team accepted the risk of this finding. They also stated that will consider addressing this issue in future versions of the protocol.

3.14 (HAL-14) PRIVILEGED ADDRESS CAN BE TRANSFERRED WITHOUT CONFIRMATION - LOW

Description:

An incorrect use of the execute_update_config function in contracts can set owner to an invalid address and inadvertently lose control of the contracts, which cannot be undone in any way. Currently, the owner of the contracts can change **owner address** using the aforementioned function in a single transaction and without confirmation from the new address.

It is worth noting that likelihood for this to happen is low because most contracts are intended to be owned by governance indefinitely, who is the responsible one for these operations. The affected smart contracts are the following:

- mars-safety-fund
- mars-treasury
- mars-address-provider
- mars-incentives
- mars-protocol-rewards-collector
- mars-oracle
- mars-red-bank
- mars-staking

Code Location:

Listing 19: contracts/mars-safety-fund/src/contract.rs (Line 80)

```
76 if info.sender != config.owner {
77     return Err(MarsError::Unauthorized {});
78 };
79
80 config.owner = option_string_to_addr(deps.api, owner, config.owner
    )?;
```

Listing 20: contracts/mars-treasury/src/contract.rs (Line 80)

```
76 if info.sender != config.owner {
77     return Err(MarsError::Unauthorized {});
78 };
79
80 config.owner = option_string_to_addr(deps.api, owner, config.owner
       )?;
```

Listing 21: contracts/mars-address-provider/src/contract.rs (Line 89)

```
68     if info.sender != config.owner {
69         return Err(ContractError::Unauthorized {});
70     }
71
72     let ConfigParams {
73         owner,
74         council_address,
75         incentives_address,
76         safety_fund_address,
77         mars_token_address,
78         oracle_address,
79         protocol_admin_address,
80         protocol_rewards_collector_address,
81         red_bank_address,
82         staking_address,
83         treasury_address,
84         vesting_address,
85         xmars_token_address,
86     } = config_params;
87
88     // Update config
89     config.owner = option_string_to_addr(deps.api, owner, config.
           owner)?;
```

Listing 22: contracts/mars-incentives/src/contract.rs (Line 281)

```
277     if info.sender != config.owner {
278         return Err(MarsError::Unauthorized {});
279     };
280
281     config.owner = option_string_to_addr(deps.api, owner, config.
           owner)?;
```

Listing 23: contracts/mars-protocol-rewards-collector/src/contract.rs (Line 136)

```
121     if info.sender != config.owner {
122         return Err(MarsError::Unauthorized {}.into());
123     }
124
125     // Destructuring a structs fields into separate variables in
            order to force
126     // compile error if we add more params
127     let CreateOrUpdateConfig {
128         owner,
129         address_provider_address,
130         safety_fund_fee_share,
131         treasury_fee_share,
132         astroport_factory_address,
133         astroport_max_spread,
134     } = new_config;
135
136     config.owner = option_string_to_addr(deps.api, owner, config.
            owner)?;
```

Listing 24: contracts/mars-oracle/src/contract.rs (Line 67)

```
63     if info.sender != config.owner {
64         return Err(MarsError::Unauthorized {}.into());
65     };
66
67     config.owner = option_string_to_addr(deps.api, owner, config.
            owner)?;
```

Listing 25: contracts/mars-red-bank/src/contract.rs (Line 293)

```
279     if info.sender != config.owner {
280         return Err(MarsError::Unauthorized {}.into());
281     }
282
283     // Destructuring a structs fields into separate variables in
            order to force
284     // compile error if we add more params
285     let CreateOrUpdateConfig {
286         owner,
287         address_provider_address,
```

```
288             ma_token_code_id,
289             close_factor,
290         } = new_config;
291
292         // Update config
293         config.owner = option_string_to_addr(deps.api, owner, config.
                owner)?;
```

Listing 26: contracts/mars-staking/src/contract.rs (Line 159)

```
144         if info.sender != config.owner {
145             return Err(MarsError::Unauthorized {});
146         }
147
148         // Destructuring a structs fields into separate variables in
                order to force
149         // compile error if we add more params
150         let CreateOrUpdateConfig {
151             owner,
152             cooldown_duration,
153             address_provider_address,
154             astroport_factory_address,
155             astroport_max_spread,
156         } = new_config;
157
158         // Update config
159         config.owner = option_string_to_addr(deps.api, owner, config.
                owner)?;
```

Risk Level:

Likelihood - 1
Impact - 4

Recommendation:

It is recommended to split **owner transfer** functionality into set_owner and accept_ownership functions. The latter function allows the transfer to be completed by the recipient.

Remediation plan:

RISK ACCEPTED: The Mars team accepted the risk for this finding. They also stated that the overhead of **Council** having to accept ownership wouldn't be as beneficial as the potential risk it mitigates.

3.15 (HAL-15) USERS CAN BURN THEIR OWN XMARS TOKENS - LOW

Description:

execute_burn function in **contracts/mars-xmars-token/src/contract.rs** is not restricted enough. As a consequence, an attacker could use external methods, such as phishing, to fool legitimate users to burn their own xMars tokens.

Code Location:

Listing 27: contracts/mars-xmars-token/src/contract.rs (Line 144)

```
138 pub fn execute_burn(
139     deps: DepsMut,
140     env: Env,
141     info: MessageInfo,
142     amount: Uint128,
143 ) -> Result<Response, ContractError> {
144     core::burn(deps.storage, &env, &info.sender, amount)?;
145
146     let res = Response::new()
147         .add_attribute("action", "burn")
148         .add_attribute("user", info.sender)
149         .add_attribute("amount", amount);
150     Ok(res)
151 }
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

It is recommended that xMars token owners cannot call the execute_burn function. This type of function should be restricted to the appropriate authority, e.g.: **mars-staking** contract.

Remediation plan:

RISK ACCEPTED: The Mars team accepted the risk of this finding. They also claimed that phishing can be used to get users to transfer xMars tokens anyway.

3.16 (HAL-16) QUERYING PRICE COULD PANIC FOR ASSETS WITH TWAP PRICE SOURCES - LOW

Description:

query_asset_price function in **contracts/mars-oracle/src/contract.rs** could panic for assets with **AstroportTwap** as price source if a transaction in the same block has previously called execute_record_twap_snapshots function.

This issue happens because of a **division by 0** when last snapshot is registered with timestamp = **env.block.time.senconds()**.

Code Location:

In block "N", execute_record_twap_snapshots function is called and timestamp has **env.block.time.senconds()** as a value.

Listing 28: contracts/mars-oracle/src/contract.rs (Line 114)

```
108 pub fn execute_record_twap_snapshots(
109     deps: DepsMut,
110     env: Env,
111     _info: MessageInfo,
112     assets: Vec<Asset>,
113 ) -> Result<Response, ContractError> {
114     let timestamp = env.block.time.seconds();
```

ASTROPORT_TWAP_SNAPSHOTS stores the snapshot with the current timestamp.

Listing 29: contracts/mars-oracle/src/contract.rs (Lines 156-157,161)

```
156 snapshots.push(AstroportTwapSnapshot {
157     timestamp,
158     price_cumulative,
159 });
```

```
160
161 ASTROPORT_TWAP_SNAPSHOTS.save(deps.storage, &asset_reference, &
        snapshots)?;
```

In a later transaction in block “N”, query_asset_price function is called for an asset with **AstroportTwap** as price source. Current snapshot has timestamp with **env.block.time.senconds()** as a value.

Listing 30: contracts/mars-oracle/src/contract.rs (Lines 248,251-252)

```
243 PriceSourceChecked::AstroportTwap {
244     pair_address,
245     window_size,
246     tolerance,
247 } => {
248     let snapshots = ASTROPORT_TWAP_SNAPSHOTS.load(deps.storage, &
            asset_reference)?;
249
250     // First, query the current TWAP snapshot
251     let current_snapshot = AstroportTwapSnapshot {
252         timestamp: env.block.time.seconds(),
253         price_cumulative: query_astroport_cumulative_price(&deps.
                querier, &pair_address)?,
254     };
```

period becomes zero because timestamp for current and previous snapshots have the same value. Finally, the transaction will panic because of the **division by zero** operation.

Listing 31: contracts/mars-oracle/src/contract.rs (Line 278)

```
277 let period = current_snapshot.timestamp - previous_snapshot.
        timestamp;
278 let price = Decimal::from_ratio(price_delta, period);
279
280 Ok(price)
```

Risk Level:

Likelihood - 3
Impact - 2

Recommendation:

Update the logic of `query_asset_price` function to handle correctly the case where a transaction in the same block has previously called `execute_record_twap_snapshots` function.

Remediation plan:

RISK ACCEPTED: The Mars team accepted the risk of this finding.

3.17 (HAL-17) ROUNDING ISSUES WHEN DISTRIBUTING REWARDS TO PROTOCOL CONTRACTS - LOW

Description:

execute_distribute_protocol_rewards function in **contracts/mars-protocol-rewards-collector/src/contract.rs** has rounding issues when calculating the values of safety_fund_amount and treasury_amount. In some scenarios, their values can be rounded down even to zero, which could affect the rewards' distribution to **mars-safety-fund** and **mars-treasury** contracts.

The risk level for this finding increases because the aforementioned function can be called by malicious users at anytime, using a specific amount to generate the rounding issues.

Code Location:

Listing 32: contracts/mars-protocol-rewards-collector/src/contract.rs (Lines 265,266,269-270)

```
265 let safety_fund_amount = amount_to_distribute * config.
    safety_fund_fee_share;
266 let treasury_amount = amount_to_distribute * config.
    treasury_fee_share;
267 let amount_to_distribute_before_staking_rewards =
268     safety_fund_amount.checked_add(treasury_amount)?;
269 let staking_amount =
270     amount_to_distribute.checked_sub(
        amount_to_distribute_before_staking_rewards)?;
```

Risk Level:

Likelihood - 3
Impact - 2

Recommendation:

Update the logic of execute_distribute_protocol_rewards function to throw error messages when the value of safety_fund_amount or treasury_amount is zero.

Remediation plan:

RISK ACCEPTED: The Mars team accepted the risk of this finding. They also stated that, in their opinion, there is no apparent risk of loss of funds because unsent funds would remain with the rewards distributor.

3.18 (HAL-18) LIQUIDATION THRESHOLD VALUE CAN BE CHANGED UNRESTRICTEDLY - LOW

Description:

execute_update_asset function in **contracts/mars-red-bank/src/contract.rs** allows the unrestricted modification of liquidation_threshold value. If mistakenly done, it would imply that many debt positions can be liquidated in unfair fashion, which severely affects borrowers' lending strategy.

It is worth noting that likelihood for this to happen is low because **mars-red-bank** contract is intended to be owned by governance indefinitely, who is the responsible one for this operation.

Code Location:

execute_update_asset function updates the value of liquidation_threshold and calls validate function to verify if this new value is appropriate.

Listing 33: contracts/mars-red-bank/src/contract.rs (Lines 570,584)

```
567 let mut updated_market = Market {
568     max_loan_to_value: max_loan_to_value.unwrap_or(market.
        max_loan_to_value),
569     reserve_factor: reserve_factor.unwrap_or(market.reserve_factor
        ),
570     liquidation_threshold: liquidation_threshold
571         .unwrap_or(market.liquidation_threshold),
572     liquidation_bonus: liquidation_bonus.unwrap_or(market.
        liquidation_bonus),
573     active: active.unwrap_or(market.active),
574     deposit_enabled: deposit_enabled.unwrap_or(market.
        deposit_enabled),
575     borrow_enabled: borrow_enabled.unwrap_or(market.borrow_enabled
        ),
576     ..market
577 };
```

```
578
579 if let Some(params) = interest_rate_model_params {
580     updated_market.interest_rate_model =
581         init_interest_rate_model(params, env.block.time.seconds())
                ?;
582 }
583
584 updated_market.validate()?;
```

validate function verifies if the value of liquidation_threshold is lesser or equal than 1 and also greater than max_loan_to_value. However, it does not verify if the new value has a significant difference with the previous one.

Listing 34: packages/mars-core/src/red_bank/mod.rs (Lines 106,117)

```
 97 pub fn validate(&self) -> Result<(), MarketError> {
 98     // max_loan_to_value, reserve_factor, liquidation_threshold
           and liquidation_bonus should be less or equal 1
 99     let conditions_and_names = vec![
100         (
101             self.max_loan_to_value.le(&Decimal::one()),
102             "max_loan_to_value",
103         ),
104         (self.reserve_factor.le(&Decimal::one()), "reserve_factor"
               ),
105         (
106             self.liquidation_threshold.le(&Decimal::one()),
107             "liquidation_threshold",
108         ),
109         (
110             self.liquidation_bonus.le(&Decimal::one()),
111             "liquidation_bonus",
112         ),
113     ];
114     all_conditions_valid(conditions_and_names)?;
115
116     // liquidation_threshold should be greater than
           max_loan_to_value
117     if self.liquidation_threshold <= self.max_loan_to_value {
118         return Err(MarketError::InvalidLiquidationThreshold {
119             liquidation_threshold: self.liquidation_threshold,
120             max_loan_to_value: self.max_loan_to_value,
```

```
121             });
122         }
123
124         Ok(())
125     }
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

Update the logic of execute_update_asset function to include a **ramp change schema** for liquidation_threshold that includes the following criteria:

- Minimum time window between changes.
- New value cannot be lower than a predefined threshold.
- New value should not differ more than a predefined amount / percentage from the previous one.

As a reference, **ramp change schema** for Curve protocol is included in the following link.

Remediation plan:

RISK ACCEPTED: The Mars team accepted the risk of this finding. They also stated that this risk should be considered by the community when proposing new threshold values.

3.19 (HAL-19) REPEATED ASSETS CAN BE INITIALIZED - LOW

Description:

execute_init_asset function in **contracts/mars-red-bank/src/contract.rs** allows the possibility to create markets with already existing assets, which generates unexpected situations, e.g.: indirectly reducing accrued protocol rewards. This issue happens because **get_attributes** function will return different values if two addresses differ just in their upper / lower cases.

It is worth noting that likelihood for this to happen is low because **mars-red-bank** contract is intended to be owned by governance indefinitely, who is the responsible one for this operation.

Code Location:

execute_init_asset function calls get_attributes function for the **asset** introduced as a parameter.

Listing 35: contracts/mars-red-bank/src/contract.rs (Line 329)

```
327 let mut money_market = GLOBAL_STATE.load(deps.storage)?;
328
329 let (asset_label, asset_reference, asset_type) = asset.
      get_attributes();
330 let market_option = MARKETS.may_load(deps.storage, asset_reference
      .as_slice())?;
331 match market_option {
332     None => {
333         let market_idx = money_market.market_count;
334         let new_market = create_market(
335             env.block.time.seconds(),
336             market_idx,
337             asset_type,
338             asset_params,
339         )?;
```

get_attributes function will return different values depending if **contract_addr** uses upper or lower cases.

Listing 36: packages/mars-core/src/asset.rs (Lines 30-31)

```
23 pub fn get_attributes(&self) -> (String, Vec<u8>, AssetType) {
24     match &self {
25         Asset::Native { denom } => {
26             let asset_reference = denom.as_bytes().to_vec();
27             (denom.to_string(), asset_reference, AssetType::Native)
28         }
29         Asset::Cw20 { contract_addr } => {
30             let asset_reference = contract_addr.as_bytes().to_vec();
31             (contract_addr.to_string(), asset_reference, AssetType::
                   Cw20)
32         }
33     }
34 }
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

Update the logic of execute_init_asset function to turn addresses in Asset into lowercase before calling **get_attributes** function.

Remediation plan:

SOLVED: The issue was fixed in commit 5a0acf1ffe06ef81a54b190469429b30c47630ed.

3.20 (HAL-20) SOME QUERY MESSAGES DO NOT HANDLE ERRORS ADEQUATELY - LOW

Description:

Some query messages in **contracts/mars-red-bank/src/contract.rs** do not handle errors adequately and will always (cannot be undone) throw error messages if a failed asset has been mistakenly initialized previously. The affected query messages are the following:

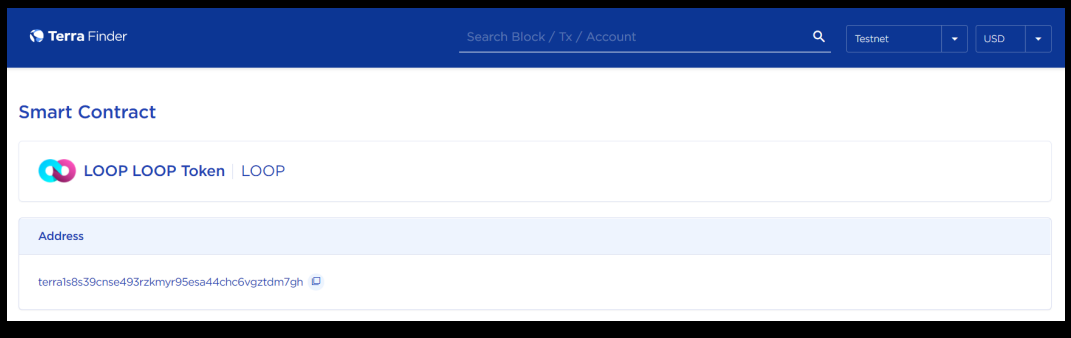
- `QueryMsg::MarketsList`
- `QueryMsg::UserDebt`
- `QueryMsg::UserCollateral`

This issue happens because all those messages mentioned above calls `get_asset_identifiers` function, which throws error messages when trying to get a symbol from a failed CW20 contract address.

It is worth noting that the likelihood for this to happen is low because **mars-red-bank** contract is intended to be owned by governance indefinitely, who is the responsible for initializing assets.

Proof of concept:

1. For the example, a failed LOOP LOOP token deployed in testnet will be initialized as asset. The address of the token is **terra1s8s39cnse493rzkmyr95esa44chc6vgztdm7gh**:



2. QueryMsg::MarketsList query message is invoked:

```
let response = await terra.wasm.contractQuery(
    red_bank_address,
    { markets_list: { } }
  );

console.log(response);
```

3. The query message will always throw error messages because fails in getting the symbol from CW20 contract address mentioned in **Step 1**:

```
    data: {
      error: 'rpc error: code = Unknown desc = Generic error: failed to get symbol from cw20
contract address: TERRA1S8S39CNSE493RZKMYR95ESA44CHC6VGZTDM7GH: contract query failed'
    }
```

Code Location:

Listing 37: contracts/mars-red-bank/src/contract.rs (Line 1891)

```
1885 pub fn query_markets_list(deps: Deps) -> StdResult<
       MarketsListResponse> {
1886     let markets_list: StdResult<Vec<_>> = MARKETS
1887         .range(deps.storage, None, None, Order::Ascending)
1888         .map(|item| {
1889             let (asset_reference, market) = item?;
1890             let (denom, asset_label) =
1891                 get_asset_identifiers(deps, asset_reference.clone
                       (), market.asset_type)?;
```

Listing 38: contracts/mars-red-bank/src/contract.rs (Line 1918)

```
1908 pub fn query_user_debt(deps: Deps, env: Env, user_address: Addr)
       -> StdResult<UserDebtResponse> {
1909     let user = USERS
1910         .may_load(deps.storage, &user_address)?
1911         .unwrap_or_default();
1912
1913     let debts: StdResult<Vec<_>> = MARKETS
1914         .range(deps.storage, None, None, Order::Ascending)
1915         .map(|item| {
1916             let (asset_reference, market) = item?;
```

```
917             let (denom, asset_label) =
918                 get_asset_identifiers(deps, asset_reference.clone
                        (), market.asset_type)?;
```

Listing 39: contracts/mars-red-bank/src/contract.rs (Line 1987)

```
1979 pub fn query_user_collateral(deps: Deps, address: Addr) ->
       StdResult<UserCollateralResponse> {
1980     let user = USERS.may_load(deps.storage, &address)?.
           unwrap_or_default();
1981
1982     let collateral: StdResult<Vec<_>> = MARKETS
1983         .range(deps.storage, None, None, Order::Ascending)
1984         .map(|item| {
1985             let (asset_reference, market) = item?;
1986             let (denom, asset_label) =
1987                 get_asset_identifiers(deps, asset_reference.clone
                        (), market.asset_type)?;
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

Update the logic of functions mentioned above to ignore error messages from failed assets (if any) and show accurate information for the other assets.

Remediation plan:

RISK ACCEPTED: The Mars team accepted the risk of this finding. They also claimed that it is better for them to take a chance here and let the query fail. If the wrong asset is initialized, they would have to address it at the protocol level, plus the queries just fail.

3.21 (HAL-21) FUNCTION TO SWAP ASSET TO UUSD IS NOT RESTRICTED ENOUGH - LOW

Description:

execute_swap_asset_to_uusd function in **contracts/mars-protocol-rewards-collector/src/contract.rs** is not restricted enough. As a consequence, a malicious user could call this function when **asset-UST** Astroport pool is imbalanced or in other adverse circumstances, which could affect negatively funds in **mars-protocol-rewards-collector** contract.

It is worth noting that impact for this issue is limited by astroport_max_spread during swapping.

Code Location:

Listing 40: contracts/mars-protocol-rewards-collector/src/contract.rs (Lines 335-343)

```
321 pub fn execute_swap_asset_to_uusd(
322     deps: DepsMut,
323     env: Env,
324     offer_asset_info: AssetInfo,
325     amount: Option<Uint128>,
326 ) -> StdResult<Response> {
327     let config = CONFIG.load(deps.storage)?;
328
329     let ask_asset_info = AssetInfo::NativeToken {
330         denom: "uusd".to_string(),
331     };
332
333     let astroport_max_spread = Some(config.astroport_max_spread);
334
335     execute_swap(
336         deps,
337         env,
338         offer_asset_info,
```

```
339             ask_asset_info,
340             amount,
341             config.astroport_factory_address,
342             astroport_max_spread,
343         )
344 }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is recommended to restrict access to execute_swap_asset_to_uusd function to the appropriate authority, e.g.: **Mars-controlled bot**.

Remediation plan:

PENDING: The Mars team stated that they are aware of this issue and are planning ways to mitigate it in future versions of the protocol.

3.22 (HAL-22) POTENTIAL ASSET PRICE OVERWRITING - INFORMATIONAL

Description:

`execute_set_asset` function in **contracts/mars-oracle/src/contract.rs** allows owner to set the price source (which impacts in the asset price) for new assets and existing ones as well. For the latter ones, the function does not validate if an asset has a price source already assigned, which means the owner could mistakenly send the same message to overwrite an existing price source.

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

Code Location:

Listing 41: contracts/mars-oracle/src/contract.rs (Lines 86-88)

```
74 pub fn execute_set_asset(
75     deps: DepsMut,
76     _env: Env,
77     info: MessageInfo,
78     asset: Asset,
79     price_source_unchecked: PriceSourceUnchecked,
80 ) -> Result<Response, ContractError> {
81   let config = CONFIG.load(deps.storage)?;
82   if info.sender != config.owner {
83       return Err(MarsError::Unauthorized {}.into());
84   }
85
86   let (asset_label, asset_reference, _) = asset.get_attributes();
87   let price_source = price_source_unchecked.to_checked(deps.api)?;
88   PRICE_SOURCES.save(deps.storage, &asset_reference, &price_source
       )?;
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

Verify if an asset has a price source already assigned in `execute_set_asset` and implement a separate function to explicitly update price sources for existing assets.

Remediation plan:

ACKNOWLEDGED: The Mars team acknowledged this finding.

3.23 (HAL-23) OPTIMAL UTILIZATION RATE COULD SKIP VALIDATION OF AN INADEQUATE VALUE - INFORMATIONAL

Description:

validate functions in **packages/mars-core/src/red_bank/interest_rate_models.rs** allow that optimal_utilization_rate value is **Decimal::one()**, whereby operations could panic when calculating **borrow rate** in update_market_interest_rates_with_model function.

It is worth noting that this is an unlikely scenario because optimal_utilization_rate is restricted by current_utilization_rate value, which is lower than **Decimal::one()**.

Code Location:

Listing 42: packages/mars-core/src/red_bank/interest_rate_models.rs (Line 176)

```
168 pub fn validate(&self) -> Result<(), InterestRateModelError> {
169     if self.min_borrow_rate > self.max_borrow_rate {
170         return Err(InterestRateModelError::InvalidMinMaxBorrowRate
                {
171             min_borrow_rate: self.min_borrow_rate,
172             max_borrow_rate: self.max_borrow_rate,
173         });
174     }
175
176     if self.optimal_utilization_rate > Decimal::one() {
177         return Err(InterestRateModelError::
               InvalidOptimalUtilizationRate {});
178     }
179
180     Ok(())
181 }
```

Listing 43: packages/mars-core/src/red_bank/interest_rate_models.rs (Line 258)

```
257 pub fn validate(&self) -> Result<(), InterestRateModelError> {
258     if self.optimal_utilization_rate > Decimal::one() {
259         return Err(InterestRateModelError::
                InvalidOptimalUtilizationRate {});
260     }
261
262     Ok(())
263 }
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

Update the logic of validate functions to throw an error message when optimal_utilization_rate is greater or equal than **Decimal::one()**.

Remediation plan:

ACKNOWLEDGED: The Mars team acknowledged this finding.

3.24 (HAL-24) UPDATED BORROW INDEX COULD BE WRONGLY CALCULATED - INFORMATIONAL

Description:

get_updated_borrow_index function in **contracts/mars-red-bank/src/interest_rates.rs** does not restrict that indexes_last_updated is lower or equal than **(current) timestamp**, whereby borrow_index value could be wrongly calculated because the function will return the current value of borrow index when timestamp has a past value.

It is worth noting that this is an unlikely scenario because timestamp will have **env.block.time.seconds()** value when users try to borrow, repay or liquidate.

Code Location:

Listing 44: contracts/mars-red-bank/src/interest_rates.rs (Line 204)

```
203 pub fn get_updated_borrow_index(market: &Market, timestamp: u64)
        -> StdResult<Decimal> {
204     if market.indexes_last_updated < timestamp {
205         let time_elapsed = timestamp - market.indexes_last_updated;
206
207         if market.borrow_rate > Decimal::zero() {
208             let updated_index =
                    calculate_applied_linear_interest_rate(
209                 market.borrow_index,
210                 market.borrow_rate,
211                 time_elapsed,
212             );
213             return updated_index;
214         }
215     }
216
217     Ok(market.borrow_index)
218 }
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

Update the logic of get_updated_borrow_index function to throw an error message when indexes_last_updated is greater than **timestamp**.

Remediation plan:

ACKNOWLEDGED: The Mars team acknowledged this finding.

3.25 (HAL-25) INACCURATE ERROR MESSAGES - INFORMATIONAL

Description:

Error messages shown in certain sections of code have inaccurate information, which could mislead legitimate users if these messages appear during a failed operation with the Mars protocol.

Code Location:

Error message when asset is already initialized is not accurate.

Listing 45: contracts/mars-red-bank/src/error.rs (Line 49)

```
49     #[error("User's health factor can't be less than 1 after
           withdraw")]
50     AssetAlreadyInitialized {},
```

Error message when liquidation is not possible because of positive uncollateralized loan limit is not accurate.

Listing 46: contracts/mars-red-bank/src/error.rs (Line 82)

```
82     #[error("Amount to repay is greater than total debt")]
83     CannotLiquidateWhenPositiveUncollateralizedLoanLimit {},
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Correct error messages to show more accurate information and to avoid confusing users if these messages appear.

Remediation plan:

SOLVED: The issue was fixed in the following commits:

- 5b28e1a634e17f8b2b5c58324deb9638d98dc58d
- cfa6954a4987cba4880ea12102e27c2a3208f4f7

3.26 (HAL-26) MISUSE OF HELPER METHODS - INFORMATIONAL

Description:

The use of the unwrap and expect 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 or expect functions in production environments is a bad practice because not only will this cause the program to crash out, or panic!, but also (in case of unwrap) no helpful messages are shown to help the user solve, or understand the reason of the error.

The affected smart contracts are the following:

- mars-vesting
- mars-incentives
- mars-protocol-rewards-collector

Code Location:

Listing 47: Resources affected

```
contracts/mars-vesting/src/contract.rs#89:    let
    mars_token_address = addresses_query.pop().unwrap();
contracts/mars-vesting/src/contract.rs#90:    let staking_address
    = addresses_query.pop().unwrap();
contracts/mars-vesting/src/contract.rs#91:    let
    protocol_admin_address = addresses_query.pop().unwrap();
contracts/mars-vesting/src/contract.rs#142:    let
    xmars_token_address = addresses_query.pop().unwrap();
contracts/mars-vesting/src/contract.rs#143:    let staking_address
     = addresses_query.pop().unwrap();
contracts/mars-vesting/src/contract.rs#241:        0 =>
    after_staking(deps, env, reply.result.unwrap().events),
contracts/mars-incentives/src/contract.rs#243:        let
    staking_address = addresses_query.pop().unwrap();,
contracts/mars-incentives/src/contract.rs#244:       let
    mars_token_address = addresses_query.pop().unwrap();,
```

```
 9 contracts/mars-protocol-rewards-collector/src/contract.rs#60
      safety_fund_fee_share: safety_fund_fee_share.unwrap(),
10 contracts/mars-protocol-rewards-collector/src/contract.rs#61
      treasury_fee_share: treasury_fee_share.unwrap(),
11 contracts/mars-protocol-rewards-collector/src/contract.rs#67
      astroport_max_spread: astroport_max_spread.unwrap(),
12 contracts/mars-protocol-rewards-collector/src/contract.rs#261
            let treasury_address = addresses_query.pop().unwrap();
13 contracts/mars-protocol-rewards-collector/src/contract.rs#262
            let staking_address = addresses_query.pop().unwrap();
14 contracts/mars-protocol-rewards-collector/src/contract.rs#263
            let safety_fund_address = addresses_query.pop().unwrap();
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to not use the unwrap or expect functions in a production environment because this use provokes panic! and may crash the Spectrum contracts 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:

ACKNOWLEDGED: The Mars team acknowledged this finding.

3.27 (HAL-27) 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 contract and package, which could lead to unexpected consequences if the project is refactored.

Code Location:

Listing 48: Resources affected

```
contracts/mars-address-provider/Cargo.toml
contracts/mars-council/Cargo.toml
contracts/mars-ma-token/Cargo.toml
contracts/mars-incentives/Cargo.toml
contracts/mars-oracle/Cargo.toml
contracts/mars-protocol-rewards-collector/Cargo.toml
contracts/mars-red-bank/Cargo.toml
contracts/mars-safety-fund/Cargo.toml
contracts/mars-staking/Cargo.toml
contracts/mars-treasury/Cargo.toml
contracts/mars-vesting/Cargo.toml
contracts/mars-xmars-token/Cargo.toml
packages/mars-core/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:

SOLVED: The issue was fixed in commit 3bd78b3335a7ec19308ae7b0088e3ab25125e01e.

3.28 (HAL-28) MULTIPLE INSTANCES OF UNCHECKED ARITHMETIC - INFORMATIONAL

Description:

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

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 49: Resources affected

```
mars-vesting: contract.rs (#L151, 175, 181, 225, 228, 229, 367, 378, 387)
mars-incentives: contract.rs (#L357, 358, 374)
mars-protocol-rewards-collector: contract.rs (#L265, 266, 310)
mars-red-bank: contract.rs (#L835, 839, 989, 1145, 1586, 1596, 1598, 1610)
mars-red-bank: accounts.rs (#L86, 90, 91, 94)
mars-ma-token: core.rs (#L85)
```

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:

PARTIALLY SOLVED: Commit d3e9e36b4dea5b66d636e5e52bf29293261f6d3b fixes the security issue for the following resources:

Listing 50: Fixed resources

```
1 mars-incentives: contract.rs (#L358, 374)
2 mars-red-bank: contract.rs (#L1610)
```

The Mars team also claimed that to add verified arithmetic for the remaining cases, they would need to add custom methods which they would prefer to avoid at this stage, but which the community could consider in future versions of the protocol.

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. All vulnerabilities shown here were already disclosed in the above report. However, 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 warnings:
- const-oid 0.6.0 is yanked
- crypto-bigint 0.2.2 is yanked

THANK YOU FOR CHOOSING

// HALBORN