Audit Report

Mars Outposts

v0.5

October 28, 2022

Table of Contents

License

Disclaimer

This audit has been performed by

Oak Security

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security has been engaged by Delphi Labs Ltd. to perform a security audit of the Mars Outposts smart contracts.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following GitHub repository:

https://github.com/mars-protocol/outposts

Commit hash: `62666cc07627ff41acda7196ca34ad8dbc1f1f8d`

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line by line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

The submitted contracts implement Mars, a money market protocol built on the Osmosis blockchain.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending,** **Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Summary of Findings

No	Description	Severity	Status
1	Disabled collateral can be re-enabled by depositing on behalf of the user	Critical	Resolved
2	Swapping assets in the reward collector contract are vulnerable to sandwich attack	Major	Resolved
3	Red bank's markets with an id greater than 128 cannot be used	Major	Resolved
4	Incorrect refund address during debt repayment	Major	Resolved
5	Computationally heavy unbounded loop during user position health calculation can lead to out-of-gas execution	Minor	Acknowledged
6	Address provider contract does not validate the newly set owner as valid address	Minor	Resolved
7	Users will be unable to claim rewards if too many asset incentives are added	Minor	Acknowledged
8	Addresses are not validated if a wrong `prefix` is set in the `address-provider` contract	Minor	Resolved
9	Setting price source for liquidity token affects `Prices` query message	Minor	Resolved
10	`Market`'s `reserve_factor` attribute is not validated	Minor	Resolved
11	Contracts are not compliant with CW2 Migration specification	Minor	Resolved
12	Unversioned dependencies could lead to supply chain attacks	Minor	Resolved
13	`optimal_utilization_ratio` value could cause a division by zero panic if not correctly validated	Minor	Resolved
14	Zero Mars token code identifier will lead to `InitAsset` failures	Minor	Resolved
15	Liquidator is unable to specify the receiver address which negatively impacts integrations and flexibility	Informational	Resolved

16	Avoid meaningless configuration update	Informational	Resolved
17	`InvalidDepositAmount` validation during red bank deposit can be removed	Informational	Resolved
18	Overflow checks not enabled for release profile	Informational	Resolved
19	Contracts should implement a two step ownership transfer	Informational	Acknowledged
20	Exponential TWAP should be used instead of arithmetic TWAP	Informational	Acknowledged
21	Custom access controls implementation	Informational	Acknowledged

Code Quality Criteria

Criteria	Status	Comment
Code complexity	Medium-High	-
Code readability and clarity	Medium-High	-
Level of documentation	High	The Mars team provided sufficient documentation including flow diagrams and code walk-throughs.
Test coverage	High	`cargo tarpaulin` reports 94.31% code coverage.

Detailed Findings

1. Disabled collateral can be re-enabled by depositing on behalf of the user

Severity: Critical

In `contracts/red-bank/src/execute.rs:471`, the user's collateral to market bit is automatically set if the user deposits a collateral asset. An attacker can re-enable a disabled asset as collateral by depositing a small number of funds on behalf of a victim. As a result, the re-enabled collateral will get liquidated if the borrower does not maintain the sufficient liquidation threshold ratio. This might cause an unexpected loss to the victim since they disabled the asset as collateral.

Please see the `test_enable_asset_as_collateral_for_other_users` test case in the appendix to reproduce the issue.

Recommendation

We recommend enabling the asset as collateral only if the caller deposits the asset for themself. This can be done by checking whether `info.sender` is equal to the `user_addr` in `contracts/red-bank/src/execute.rs:437`.

Status: Resolved

2. Swapping assets in the reward collector contract are vulnerable to sandwich attack

Severity: Major

In `contracts/rewards-collector/osmosis/src/route.rs:121`, the minimum amount of swap output is hardcoded to zero. Due to no slippage protection, an attacker can perform a sandwich attack by purchasing the asset at the normal price, calling the `SwapAsset` operation to buy the asset at an increased price, then immediately selling it for a profit. The attacker can repeatedly perform this attack to force the contract into buying assets at a higher price, resulting in a loss of funds.

Recommendation

We recommend resolving the TODO comment and implementing slippage protection.

Status: Resolved

This issue has been addressed using a 10-minute TWAP to calculate the correct output amount. This solution requires an unbounded and gas-intensive iteration through all swap

steps in order to perform TWAP queries, but routes can be overwritten to prevent running out of gas.

3. Red bank's markets with an id greater than 128 cannot be used

Severity: Major

The red bank contract can handle an indefinite amount of `Markets` that are created by the contract owner and stored in the `MARKETS` mapping.

Each `Market` struct has a unique `index` attribute that is responsible for indicating its bit position in `borrowed_assets` and `collateral_assets` bitmaps defined in the `User` struct.

Since those bitmaps are `Uint128`, they can handle a maximum of 128 `Markets`.

This implies that any `Market` with an `index` greater than 128 cannot be used because of an overflow in `contracts/red-bank/src/helpers.rs:29`.

Recommendation

We recommend using a bigger sized type for bitmaps, for example `Uint512`, and implementing a hard cap and a check in order to prevent creating `Markets` with an `index` greater than the bitmap size.

Status: Resolved

4. Incorrect refund address during debt repayment

Severity: Major

The repay function allows repayment of a loan on behalf of other accounts. In a scenario where the caller overpays debt on behalf of another account in `contracts/red-bank/src/execute.rs:843`, the function will refund the excess amount to the target address passed in the `on_behalf_of` parameter and not the caller. This implies that the party repaying the loan will lose funds to the debtor.

Recommendation

We recommend returning all excess amounts to the payer and not to the account on whose behalf the loan is being repaid.

Status: Resolved

5. Computationally heavy unbounded loop during user position health calculation can lead to out-of-gas execution

Severity: Minor

In `contracts/red-bank/src/health.rs:121`, the `get_user_positions` functionality attempts to loop over all markets initialized in the red bank contract. Since the market count is unbounded and markets cannot be removed, the execution gets more expensive with the number of markets and might eventually run out of gas if there are too many markets initialized.

This issue is also present in the `UserDebts` and `UserCollaterals` query messages in `contracts/red-bank/src/query.rs:123-149` and `164-180`. The queries attempt to loop through all markets without any pagination limit.

Recommendation

We recommend implementing a maximum limit on markets that can be initialized as suggested also in "Red bank's markets with an id greater than 128 cannot be used".

Status: Acknowledged

The Mars team states that they will not initialize many assets and that if this issue becomes a problem in practice, it will be fixed in the future.

6. Address provider contract does not validate the newly set owner as valid address

Severity: Minor

In `contracts/address-provider/src/contract.rs:77`, the contract owner is set to the `new_owner` string provided by the caller. If the new owner is not a valid address, it will cause the address provider contract to have an invalid owner. As a result, it would prevent the `SetAddress` and `TransferOwnership` functionality from working correctly.

We consider this a minor issue because only the contract owner can cause such a misconfiguration.

Recommendation

We recommend validating the `new_owner` string to be a valid address before updating the contract owner.

Status: Resolved

7. Users will be unable to claim rewards if too many asset incentives are added

Severity: Minor

In the `incentives` contract, users can call `ClaimReward` to retrieve their accrued Mars rewards.

The message is handled by the `compute_user_unclaimed_rewards` function in line `contracts/incentives/src/helpers.rs:84` which loops through all the asset incentives.

On a long enough timeframe, if many assets get added to the protocol, this gets more expensive and could eventually run out of gas and hence block claiming of rewards for a given user.

Recommendation

We recommend adding a maximum number of assets allowed or implementing a pagination logic.

Status: Acknowledged

The Mars team states that they have a small number of assets in the beginning. In the future, they can improve this function.

8. Addresses are not validated if a wrong `prefix` is set in the `address-provider` contract

Severity: Minor

In `contracts/address-provider/src/helpers.rs:11`, when the `assert_valid_addr` function is executed, all addresses that start with a `prefix` different from the stored one are assumed to be valid without executing `api.addr_validate`.

This implies that invalid addresses can be stored in the contract mapping and that the check on the given `prefix` can be bypassed.

Recommendation

We recommend enforcing that all stored addresses are validated and coherent with the stored `prefix`.

Status: Resolved

9. Setting price source for liquidity token affects `Prices` query message

Severity: Minor

In `contracts/oracle/osmosis/src/price_source.rs:125-127`, the liquidity token price source cannot be queried as it will return an `Unimplemented` error. However, the contract owner can add a liquidity token price source in lines `97-99`. This means that the `Prices` query message would be affected since it includes the liquidity token price source when querying.

Note that users can manually avoid querying the `LiquidityToken` price source by filtering out the affected entry with the `start_after` and `limit` arguments.

Please see the `test_query_liquidity_token_prices` test case in the appendix to reproduce this issue.

Recommendation

We recommend temporarily disabling the functionality of adding `LiquidityToken` as a price source until the associated query function is implemented.

Status: Resolved

10. `Market`'s `reserve_factor` attribute is not validated

Severity: Minor

In `packages/outpost/src/red_bank/market.rs:79`, the `validate` method for the `Market` struct is not checking `reserve_factor`.

Since this attribute represents the percentile of the borrow rate that is kept as protocol rewards, a value greater than one could cause the execution to distribute an incorrect amount of funds.

We consider this issue to be minor since only the owner can cause it.

Recommendation

We recommend validating the `reserve_factor` value to be in the `[0,1)` range.

Status: Resolved

11. Contracts are not compliant with CW2 Migration specification

Severity: Minor

The following contracts do not adhere to the CW2 Migration specification standard:

- `address-provider`
- `oracle`
- `incentives`
- `red-bank`
- `rewards-collector`

This may lead to unexpected problems during contract migration and code version handling.

Recommendation

We recommend following the CW2 standard in all the contracts. For reference, see https://docs.cosmwasm.com/docs/1.0/smart-contracts/migration.

Status: Resolved

12. Unversioned dependencies could lead to supply chain attacks

Severity: Minor

The `oracle` contract requires `osmo-bindings` as a dependency in `contracts/oracle/osmosis/Cargo.toml:37`.

Since it is fetching it directly from the GitHub repository without specifying the wanted commit hash or tag, any changes to that repository's main branch may accidentally be included in the contract.

This could lead to bugs as well as supply chain attacks.

Recommendation

We recommend specifying a commit or tag in `Cargo.toml` in order to fix the dependency's code version.

Status: Resolved

13. `optimal_utilization_ratio` value could cause a division by zero panic if not correctly validated

Severity: Minor

In `packages/outposts/src/red_bank/interest_rate_model.rs:35`, the `divide_decimal_by_decimal` is dividing the `current_utilization_rate` by `optimal_utilization_rate`.

Since `optimal_utilization_rate` is not validated to be greater than zero, this division could cause a panic.

Recommendation

We recommend validating `optimal_utilization_rate` to be greater than zero.

Status: Resolved

14. Zero Mars token code identifier will lead to `InitAsset` failures

Severity: Minor

In `contracts/red-bank/src/execute.rs:60`, when configuring the Mars token code identifier, it is not validated to not be zero. If the code identifier is configured as zero, it will cause the `InitAsset` functionality to fail.

This issue is also present in line `103` when updating the configuration.

Recommendation

We recommend ensuring the token identifier value is not zero in `contracts/red-bank/src/execute.rs:60` and `103`.

Status: Resolved

15. Liquidator is unable to specify the receiver address which negatively impacts integrations and flexibility

Severity: Informational

In `contracts/red-bank/src/contract.rs:72`, there is no way for the liquidator to specify the recipient address for receiving underlying collateral. This will add overhead for periphery contracts to specify the logic for redirecting the liquidated assets transfers to rightful liquidators.

Recommendation

We recommend adding a recipient parameter in `ExecuteMsg::Liquidate` to support integrations and increase flexibility.

Status: Resolved

16. Avoid meaningless configuration update

Severity: Informational

In `contracts/oracle/base/src/contract.rs:113`, the contract owner can update the configuration with the `owner` value as `None`. As a result, the execution will be meaningless because the `option_string_to_addr` functionality will default to the original configuration owner. This would cause no changes to be modified in the end.

Recommendation

We recommend changing the update configuration functionality to change the owner parameter into `String` instead of `Option<String>`.

Status: Resolved

17. `InvalidDepositAmount` validation during red bank deposit can be removed

Severity: Informational

In `contracts/red-bank/src/execute.rs:459-464`, the check ensures the deposit amount is not zero when depositing in the red bank. This check is unnecessary as the `one_coin` functionality from `cw_utils` already prevents zero amounts, as seen in `contracts/red-bank/src/contract.rs:53`.

Recommendation

We recommend removing the `InvalidDepositAmount` check to reduce gas consumption.

Status: Resolved

18. Overflow checks not enabled for release profile

Severity: Informational

The following packages and contracts do not enable `overflow-checks` for the release profile:

- `contracts/address-provider/Cargo.toml`
- `contracts/oracle/base/Cargo.toml`
- `contracts/oracle/osmosis/Cargo.toml`
- `contracts/incentives/Cargo.toml`
- `contracts/red-bank/Cargo.toml`
- `contracts/rewards-collector/base/Cargo.toml`
- `contracts/rewards-collector/osmosis/Cargo.toml`

While enabled implicitly through the workspace manifest, future refactoring might break this assumption.

Recommendation

We recommend enabling overflow checks in all packages, including those that do not currently perform calculations, to prevent unintended consequences if changes are added in future releases or during refactoring. Note that enabling overflow checks in packages other than the workspace manifest will lead to compiler warnings.

Status: Resolved

19. Contracts should implement a two step ownership transfer

Severity: Informational

The contracts within the scope of this audit allow the current owner to execute a one-step ownership transfer. While this is common practice, it presents a risk for the ownership of the contract to become lost if the owner transfers ownership to the incorrect address. A two-step ownership transfer will allow the current owner to propose a new owner, and then the account that is proposed as the new owner may call a function that will allow them to claim ownership and actually execute the config update.

Recommendation

We recommend implementing a two-step ownership transfer. The flow can be as follows:

1. The current owner proposes a new owner address that is validated.
2. The new owner account claims ownership, which applies the configuration changes.

Status: Acknowledged

The Mars team states that they may address this in a future version.

20. Exponential TWAP should be used instead of arithmetic TWAP

Severity: Informational

Arithmetic TWAP are very sensitive to price peaks. And in POS blockchains recording a price peak (using flash loan to manipulate the price) relatively needs less capital.

Consider 30 block TWAP, and attacker wants to manipulate price to 2x, and asset price is $1

1. Attackers know that he would be validating block in next epoch
2. Attackers manipulate the price to $31 by using flash loan or its own capital in the last transaction of block.number - 30. Most Dexes record spot price at the end of block
3. Attacker is now validating the next block, he will put back the price to $1 in the next block (block.number - 29)
4. Now at current Block, the TWAP is (31 + 29*1) / 30 = $2

Recommendation
we recommend using exponential TWAP instead of arithmetic TWAP

Status: Acknowledged

The Mars team mentioned that they will have such a price source in the future.

21. Custom access control implementation

Severity: Informational

Contracts implement custom access controls. Although no instances of broken controls or bypasses have been found, using a battle-tested implementation reduces potential risks and the complexity of the codebase.

Also, the access control logic is duplicated across the handlers of each function, which negatively impacts the code's readability and maintainability.

Recommendation

We recommend using a well-known access control implementation such as `cw_controllers::Admin` (https://docs.rs/cw-controllers/0.14.0/cw_controllers/struct.Admin.html).

Status: Acknowledged

The Mars team states that they plan to implement this together with the two-step ownership transfer described above.

Appendix

1. Test case for "Setting price source for liquidity token affects `Prices` query message"

```
#[test]
fn test_query_liquidity_token_prices() {
    // reproduced in contracts/oracle/osmosis/tests/test_query_price.rs

    let mut deps = helpers::setup_test();

    // normal setup
    helpers::set_price_source(
        deps.as_mut(),
        "uosmo",
        OsmosisPriceSource::Fixed {
            price: Decimal::one(),
        },
    );
    helpers::set_price_source(
        deps.as_mut(),
        "uatom",
        OsmosisPriceSource::Spot {
            pool_id: 1,
        },
    );
    helpers::set_price_source(
        deps.as_mut(),
        "umars",
        OsmosisPriceSource::Spot {
            pool_id: 89,
        },
    );

    deps.querier.set_spot_price(
        Swap {
            pool_id: 1,
            denom_in: "uatom".to_string(),
            denom_out: "uosmo".to_string(),
        },
        SpotPriceResponse {
            price: Decimal::from_ratio(77777u128, 12345u128),
        },
    );
    deps.querier.set_spot_price(
        Swap {
            pool_id: 89,
            denom_in: "umars".to_string(),
```

```
            denom_out: "uosmo".to_string(),
        },
        SpotPriceResponse {
            price: Decimal::from_ratio(88888u128, 12345u128),
        },
    );

    // normal query works
    let normal_query: Vec<PriceResponse> = helpers::query(
        deps.as_ref(),
        QueryMsg::Prices {
            start_after: None,
            limit: None,
        },
    );

    // admin set price source for liquidity token
    helpers::set_price_source(
        deps.as_mut(),
        "uoak",
        OsmosisPriceSource::LiquidityToken {
            pool_id: 12,
        },
    );

    // users are forced to avoid querying liquidity token, or else query would
fail
    let mut first_query: Vec<PriceResponse> = helpers::query(
        deps.as_ref(),
        QueryMsg::Prices {
            start_after: None,
            limit: Some(2),
        },
    );

    let mut second_query: Vec<PriceResponse> = helpers::query(
        deps.as_ref(),
        QueryMsg::Prices {
            start_after: Some("uoak".to_string()),
            limit: None,
        },
    );

    first_query.append(&mut second_query);

    // verify query results are the same
    assert_eq!(normal_query, first_query);

    // paginated query fails due to unimplemented error
    let _: Vec<PriceResponse> = helpers::query(
```

```
        deps.as_ref(),
        QueryMsg::Prices {
            start_after: None,
            limit: None,
        },
    );

}
```

2. Test case for "Disabled collateral can be re-enabled by depositing on behalf of the user"

```
#[test]
fn test_enable_asset_as_collateral_for_other_users() {
    // test case reproduced in contracts/red-bank/tests/test_deposit.rs

    // setup
    let initial_liquidity = 10000000;
    let mut deps = th_setup(&[coin(initial_liquidity, "somecoin")]);
    let ma_token_addr = Addr::unchecked("matoken");

    deps.querier.set_cw20_total_supply(ma_token_addr.clone(),
Uint128::new(10_000_000));

    let mock_market = Market {
        ma_token_address: ma_token_addr,
        liquidity_index: Decimal::one(),
        borrow_index: Decimal::one(),
        ..Default::default()
    };
    let market = th_init_market(deps.as_mut(), "somecoin", &mock_market);

    let depositor_addr = Addr::unchecked("depositor");
    let another_user_addr = Addr::unchecked("another_user");
    let deposit_amount = 110000;

    // 'another_user' deposit funds to their own account
    let env = mock_env(MockEnvParams::default());
    let info = cosmwasm_std::testing::mock_info(
        another_user_addr.as_str(),
        &[coin(deposit_amount, "somecoin")],
    );
    let msg = ExecuteMsg::Deposit {
        on_behalf_of: None,
    };
    execute(deps.as_mut(), env.clone(), info, msg).unwrap();

    // 'another_user' should have collateral bit set
    let user = USERS.load(&deps.storage, &another_user_addr).unwrap();
    assert_eq!(get_bit(user.collateral_assets, market.index).unwrap(), true);

    // 'another_user' disables asset as collateral
    let info = cosmwasm_std::testing::mock_info(
        another_user_addr.as_str(),
        &[],
    );
    let msg = ExecuteMsg::UpdateAssetCollateralStatus {
        denom: "somecoin".to_string(),
        enable: false,
```

```
    };
    execute(deps.as_mut(), env.clone(), info, msg).unwrap();

    // verify asset is disabled as collateral for 'another_user'
    let user = USERS.load(&deps.storage, &another_user_addr).unwrap();
    assert_eq!(get_bit(user.collateral_assets, market.index).unwrap(), false);

    // 'depositor' deposits a small amount of funds to 'another_user' to enable
their asset as collateral
    let env = mock_env(MockEnvParams::default());
    let info = cosmwasm_std::testing::mock_info(
        depositor_addr.as_str(),
        &[coin(1_u128, "somecoin")],
    );
    let msg = ExecuteMsg::Deposit {
        on_behalf_of: Some(another_user_addr.to_string()),
    };
    execute(deps.as_mut(), env.clone(), info, msg).unwrap();

    // 'another_user' have the asset enabled as collateral
    let user = USERS.load(&deps.storage, &another_user_addr).unwrap();
    assert_eq!(get_bit(user.collateral_assets, market.index).unwrap(), true);

}
```