// HALBORN

Mars Protocol - Outposts Rewards Collector

CosmWasm Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **September 23rd, 2022 - September 29th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	09/23/2022	Michal Bazyli
0.2	Document Update	09/27/2022	Michal Bazyli
0.3	Draft Version	09/28/2022	Michal Bazyli
0.4	Draft Review	09/30/2022	Gabi Urrutia
1.0	Remediation Plan	10/28/2022	Michal Bazyli
1.1	Remediation Plan Review	11/02/2022	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

Michal Bazyli	Halborn	Michal.Bazyli@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Mars Protocol engaged Halborn to conduct a security audit on their smart contracts beginning on September 23rd and ending on September 29th. The security assessment was scoped to the smart contracts provided in the GitHub repository Outposts, commit hashes and further details can be found in the Scope section of this report.

1.2 AUDIT SUMMARY

The team at Halborn was provided five days for the engagement and assigned a full-time security engineer to audit the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

- Ensure that smart contract functions operate as intended
- Identify potential security issues with the smart contracts

In summary, Halborn identified some security risks which were mostly addressed by the Mars Team:

- Re-implement the functions and ensure they work as intended.
- Add a safe transfer ownership logic, preferably in a form of a two-steps process.
- Enhance security of the contract by implementing validation routine when initializing or updating the config.
- Remove risky functions from the contract if there is no strong argumentation in the documentation for using such functionality.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the solidity code 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 and purpose.
- Smart contract manual code review and walkthrough.
- Manual testing by custom scripts and fuzzers.
- Scanning of Rust files for vulnerabilities, security hotspots or bugs.
- Static Analysis of security for scoped contract, and imported functions.
- Testnet deployment.

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.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

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

1.4 SCOPE

1. CosmWasm Smart Contracts
   (a) Repository: outpost
   (b) Commit ID: e9fbc50dec55f68964cf33da0f4051c0cf3d6202
   (c) Contracts in scope:
      i. rewards-collector
   (d) Packages in scope:
      i. outpost

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

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	1	0	2	1

LIKELIHOOD

IMPACT

(HAL-02)				
(HAL-03)				(HAL-01)
(HAL-04)				

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) CURRENT IMPLEMENTATION OF THE ROUTING FUNCTION DOES NOT WORK - DOS	High	SOLVED - 10/20/2022
(HAL-02) PRIVILEGED ADDRESS CAN BE TRANSFERRED WITHOUT CONFIRMATION	Low	RISK ACCEPTED
(HAL-03) MISSING VALIDATION ROUTINE FOR CRITICAL CONFIG PARAMETERS	Low	SOLVED - 10/05/2022
(HAL-04) POTENTIAL RISKY FUNCTIONS	Informational	SOLVED - 10/05/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) CURRENT IMPLEMENTATION OF THE ROUTING FUNCTION DOES NOT WORK - DOS - HIGH

Description:

set_route allow specifying swap route in Osmosis chain to swap tokens, however, the current implementation of set_route method does not work in the current state of the Osmosis chain and return error. In consequence, swap route cannot be set and swap_assets cannot be executed.

Code Location:

Implementation of set_route:

Listing 1: packages/outpost/src/rewards_collector.rs (Line 177)

```
163     fn set_route(
164         &self,
165         deps: DepsMut<Q>,
166         sender: Addr,
167         denom_in: String,
168         denom_out: String,
169         route: R,
170     ) -> ContractResult<Response<M>> {
171         let cfg = self.config.load(deps.storage)?;
172
173         if sender != cfg.owner {
174             return Err(MarsError::Unauthorized {}.into());
175         }
176
177         route.validate(&deps.querier, &denom_in, &denom_out)?;
178
179         self.routes.save(deps.storage, (denom_in.clone(),
 ↳ denom_out.clone()), &route)?;
180
181         Ok(Response::new()
182             .add_attribute("action", "mars/rewards-collector/
 ↳ set_instructions")
```

```
183             .add_attribute("denom_in", denom_in)
184             .add_attribute("denom_out", denom_out)
185             .add_attribute("route", route.to_string()))
186     }
```

Implementation of validation for route trait :

Listing 2: contracts/rewards-collector/base/src/traits.rs

```
16      fn validate(
17          &self,
18          querier: &QuerierWrapper<Q>,
19          denom_in: &str,
20          denom_out: &str,
21      ) -> ContractResult<()>;
```

Code of swap_assets:

Listing 3: contracts/rewards-collector/base/src/contract.rs

```
219     fn swap_asset(
220         &self,
221         deps: DepsMut<Q>,
222         env: Env,
223         denom: String,
224         amount: Option<Uint128>,
225     ) -> ContractResult<Response<M>> {
226         let cfg = self.config.load(deps.storage)?;
227
228         // if amount is None, swap the total balance
229         let amount_to_swap =
230             unwrap_option_amount(&deps.querier, &env.contract.
 ↳ address, &denom, amount)?;
231
232         // split the amount to swap between the safety fund and
 ↳ the fee collector
233         let amount_safety_fund = amount_to_swap * cfg.
 ↳ safety_tax_rate;
234         let amount_fee_collector = amount_to_swap.checked_sub(
 ↳ amount_safety_fund)?;
235         let mut messages = vec![];
236
```

```
237         if !amount_safety_fund.is_zero() {
238             messages.push(
239                 self.routes
240                     .load(deps.storage, (denom.clone(), cfg.
 ↳ safety_fund_denom))?
241                     .build_swap_msg(
242                         &env,
243                         &deps.querier,
244                         &denom,
245                         amount_safety_fund,
246                         cfg.slippage_tolerance,
247                     )?,
248             );
249         }
250 
251         if !amount_fee_collector.is_zero() {
252             messages.push(
253                 self.routes
254                     .load(deps.storage, (denom.clone(), cfg.
 ↳ fee_collector_denom))?
255                     .build_swap_msg(
256                         &env,
257                         &deps.querier,
258                         &denom,
259                         amount_fee_collector,
260                         cfg.slippage_tolerance,
261                     )?,
262             );
263         }
264 
265         Ok(Response::new()
266             .add_messages(messages)
267             .add_attribute("action", "outposts/rewards-collector/
 ↳ swap_asset")
268             .add_attribute("denom", denom)
269             .add_attribute("amount_safety_fund",
 ↳ amount_safety_fund)
270             .add_attribute("amount_fee_collector",
 ↳ amount_fee_collector)
271             .add_attribute("slippage_tolerance", cfg.
 ↳ slippage_tolerance.to_string()))
272     }
```

Proof of Concept:

Implementation of function which executes set_route in js:

Listing 4

```
async executeSetRoute(){
    await this.client.execute(
      this.deployerAddress,
      this.storage.addresses.rewardsCollector!,
      {
        set_route: {
          denom_in: 'uosmo',
          denom_out: 'uatom',
          route: [{ token_out_denom: this.config.atomDenom,
↳ pool_id: 1 }],
        },
      },
      'auto',
    )

  }
```

Response from the contract after executing set_route:

Listing 5

```
Error: Query failed with (18): failed to execute message; message
↳ index: 0: Error parsing into type osmosis_std::types::osmosis::
↳ gamm::v1beta1::QueryPoolResponse: missing field `type_url`:
↳ execute wasm contract failed: invalid request

```

Risk Level:

Likelihood - 5
Impact - 3

Recommendation:

It is recommended to re-implement the functionality of the methods in order to work as intended.

Remediation Plan:

SOLVED: The issue was fixed in commit 69c441c573794aca5bb5c8a6ea9dd87c3d344f7d.

3.2 (HAL-02) PRIVILEGED ADDRESS CAN BE TRANSFERRED WITHOUT CONFIRMATION - LOW

Description:

An incorrect use of the update_config function from the **rewards-collector** contract could set the owner to an invalid address, unwillingly losing control of the contract, which cannot be undone in any way. Currently, the OWNER of the contracts can change its address using the aforementioned function in a single transaction and without confirmation from the new address.

Code Location:

Listing 6: contracts/rewards-collector/base/src/contract.rss (Line 132)

```
119         fn update_config(
120         &self,
121         deps: DepsMut<Q>,
122         sender: Addr,
123         new_cfg: CreateOrUpdateConfig,
124     ) -> ContractResult<Response<M>> {
125         let mut cfg = self.config.load(deps.storage)?;
126 
127         if sender != cfg.owner {
128             return Err(MarsError::Unauthorized {}.into());
129         }
130 
131         let CreateOrUpdateConfig {
132             owner,
133             address_provider,
134             safety_tax_rate,
135             safety_fund_denom,
136             fee_collector_denom,
137             channel_id,
138             timeout_revision,
139             timeout_blocks,
```

```
140             timeout_seconds,
141             slippage_tolerance,
142         } = new_cfg;
143
144         cfg.owner = option_string_to_addr(deps.api, owner, cfg.
 ↳ owner)?;
145         cfg.address_provider =
146             option_string_to_addr(deps.api, address_provider, cfg.
 ↳ address_provider)?;
147         cfg.safety_tax_rate = safety_tax_rate.unwrap_or(cfg.
 ↳ safety_tax_rate);
148         cfg.safety_fund_denom = safety_fund_denom.unwrap_or(cfg.
 ↳ safety_fund_denom);
149         cfg.fee_collector_denom = fee_collector_denom.unwrap_or(
 ↳ cfg.fee_collector_denom);
150         cfg.channel_id = channel_id.unwrap_or(cfg.channel_id);
151         cfg.timeout_revision = timeout_revision.unwrap_or(cfg.
 ↳ timeout_revision);
152         cfg.timeout_blocks = timeout_blocks.unwrap_or(cfg.
 ↳ timeout_blocks);
153         cfg.timeout_seconds = timeout_seconds.unwrap_or(cfg.
 ↳ timeout_seconds);
154         cfg.slippage_tolerance = slippage_tolerance.unwrap_or(cfg.
 ↳ slippage_tolerance);
155
156         cfg.validate()?;
157
158         self.config.save(deps.storage, &cfg)?;
159
160         Ok(Response::new().add_attribute("action", "mars/rewards-
 ↳ collector/update_config"))
```

Risk Level:

Likelihood - 1
Impact - 4

Recommendation:

The update_config function should follow a two steps process, being split into set_owner and accept_owner functions. The latter one requiring the

transfer to be completed by the recipient, effectively protecting the contract against potential typing errors compared to single-step OWNER transfer mechanisms.

Remediation plan:

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

3.3 (HAL-03) MISSING VALIDATION ROUTINE FOR CRITICAL CONFIG PARAMETERS - LOW

Description:

The current implementation of validate function for Config struct validates only safety tax_rate is a decimal lower than 1. However, the validation routine for the maximum value of slippage_tolerance and the minimum value for timeout_* is not enforced.

The slippage_tolerance parameter is not validated against a maximum acceptable value when being set. In case when this value is set to 0.99, assets locked in the contract could be swapped with an unfavorable ratio,

Parameters, represented by timeout_*, are not validated against a minimum acceptable value when set. In case when values are set to small values or 0 it could lead to unexpected behavior or denial of service.

Code Location:

Fragment of update_config:

Listing 7: packages/outpost/src/rewards_collector.rs (Line 156)

```
119     fn update_config(
120         &self,
121         deps: DepsMut<Q>,
122         sender: Addr,
123         new_cfg: CreateOrUpdateConfig,
124     ) -> ContractResult<Response<M>> {
125         let mut cfg = self.config.load(deps.storage)?;
126
127         if sender != cfg.owner {
128             return Err(MarsError::Unauthorized {}.into());
129         }
130
131         let CreateOrUpdateConfig {
```

```
132             owner,
133             address_provider,
134             safety_tax_rate,
135             safety_fund_denom,
136             fee_collector_denom,
137             channel_id,
138             timeout_revision,
139             timeout_blocks,
140             timeout_seconds,
141             slippage_tolerance,
142         } = new_cfg;
143
144         cfg.owner = option_string_to_addr(deps.api, owner, cfg.
 ↳ owner)?;
145         cfg.address_provider =
146             option_string_to_addr(deps.api, address_provider, cfg.
 ↳ address_provider)?;
147         cfg.safety_tax_rate = safety_tax_rate.unwrap_or(cfg.
 ↳ safety_tax_rate);
148         cfg.safety_fund_denom = safety_fund_denom.unwrap_or(cfg.
 ↳ safety_fund_denom);
149         cfg.fee_collector_denom = fee_collector_denom.unwrap_or(
 ↳ cfg.fee_collector_denom);
150         cfg.channel_id = channel_id.unwrap_or(cfg.channel_id);
151         cfg.timeout_revision = timeout_revision.unwrap_or(cfg.
 ↳ timeout_revision);
152         cfg.timeout_blocks = timeout_blocks.unwrap_or(cfg.
 ↳ timeout_blocks);
153         cfg.timeout_seconds = timeout_seconds.unwrap_or(cfg.
 ↳ timeout_seconds);
154         cfg.slippage_tolerance = slippage_tolerance.unwrap_or(cfg.
 ↳ slippage_tolerance);
155
156         cfg.validate()?;
```

Fragment of `validate` impl for `Config` struct:

Listing 8: packages/outpost/src/rewards_collector.rs

```
34     impl<T> Config<T> {
35         pub fn validate(&self) -> Result<(), MarsError> {
36             decimal_param_le_one(self.safety_tax_rate, "
 ↳ safety_tax_rate")?;
```

```
37              Ok(())
38          }
39      }
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

It is recommended to add a validation routine inside `init` and `update_config` functions to ensure that:

- `slippage_tolerance` is lower than `max_slippage_tolerance`. As a reference, the maximum slippage for `Uniswap Pool` and `Uniswap Swap` is set to 50%.
- `timeout_*` is higher than the `min_timeout_*` value which should be defined in the contract

Remediation Plan:

SOLVED: The issue was fixed in commit `e3dcdc666d4fa3efb04d38b70e149abef101132c`.

3.4 (HAL-04) POTENTIAL RISKY FUNCTIONS - INFORMATIONAL

Description:

The execute_cosmos_msg allow executing CosmosMsg on behalf of the contract. However, the execute_cosmos_msg is an administrative operation, thus the likelihood that it will be exploited via such an attack vector is extremely low. However, if someone chooses to do so, any kind of CosmosMsg will be executed on behalf of the contract, e.g: Withdraw tokens from red_bank. This exposes protocol to unnecessary risk.

Code Location:

Listing 9: contracts/rewards-collector/base/src/contract.rs

```
328     fn execute_cosmos_msg(
329             &self,
330             deps: DepsMut<Q>,
331             sender: Addr,
332             msg: CosmosMsg<M>,
333         ) -> ContractResult<Response<M>> {
334             let cfg = self.config.load(deps.storage)?;
335
336             if sender != cfg.owner {
337                 return Err(MarsError::Unauthorized {}.into());
338             }
339
340             Ok(Response::new()
341                 .add_message(msg)
342                 .add_attribute("action", "outposts/rewards-
 ↳ collector/execute_cosmos_msg"))
343         }
344
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

If there is no strong argumentation in the documentation for using such functionality, removing such methods from the contract is recommended to lower the risk of exposition.

Remediation Plan:

SOLVED: The issue was fixed in commit df0a6bc952f1cae9c57703d61b3572bce8324b57.

THANK YOU FOR CHOOSING

// HALBORN