// HALBORN

Mars Protocol - Outposts Oracle

CosmWasm Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **September 19th, 2022 - November 18th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY 3
CONTACTS 4
1 EXECUTIVE OVERVIEW 5
1.1 INTRODUCTION 6
1.2 AUDIT SUMMARY 6
1.3 TEST APPROACH & METHODOLOGY 7
RISK METHODOLOGY 7
1.4 SCOPE 9
2 ASSESSMENT SUMMARY & FINDINGS OVERVIEW 11
3 FINDINGS & TECH DETAILS 12
3.1 (HAL-01) POOL PRICE SOURCE MIGHT INCREASE POTENTIAL RISKS - MEDIUM 14
Description 14
Code Location 14
Risk Level 17
Recommendation 17
Remediation plan 17
3.2 (HAL-02) PRIVILEGED ADDRESS CAN BE TRANSFERRED WITHOUT CONFIRMATION - LOW 18
Description 18
Code Location 18
Risk Level 19
Recommendation 19
Remediation plan 19
3.3 (HAL-03) FIXED PRICE SOURCE MIGHT INCREASE POTENTIAL RISKS - LOW 20

Description 20

Code Location 20

Risk Level 22

Recommendation 22

Remediation plan 22

3.4 (HAL-04) ORACLE ENTRIES CANNOT BE REMOVED - LOW 23

Description 23

Code Location 23

Risk Level 24

Recommendation 24

Remediation plan 24

3.5 (HAL-05) OVERFLOW-CHECKS BEST PRACTICES - INFORMATIONAL 25

Description 25

Code Location 25

Risk Level 25

Recommendation 25

Remediation plan 26

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	09/19/2022	Lukasz Mikula
0.2	Draft Version	09/23/2022	Lukasz Mikula
0.3	Draft Review	09/24/2022	Gabi Urrutia
1.0	Remediation Plan	10/05/2022	Lukasz Mikula
1.1	Remediation Plan Review	10/06/2022	Gabi Urrutia
1.2	Document Update	10/26/2022	Lukasz Mikula
1.3	Document Update Review	10/26/2022	Gabi Urrutia
1.4	Document Update	11/18/2022	Emiliano Carmona
1.5	Document Update Review	11/21/2022	Gabi Urrutia
1.6	Document Update	12/01/2022	Emiliano Carmona
1.7	Document Update Review	12/01/2022	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
Lukasz Mikula	Halborn	Lukasz.Mikula@halborn.com
Emiliano Carmona	Halborn	Emiliano.Carmona@halborn.com
Gustavo Dutra	Halborn	gustavo.dutra@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Mars Protocol engaged Halborn to conduct a security audit on their smart contracts beginning on September 19th and ending on November 18th. 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 assigned two full-time security engineers to audit the security of the smart contract. The security engineers 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:

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

In summary, Halborn identified some security risks that were mostly accepted by the Mars Protocol team. The main ones are the following:

- Add a safe transfer ownership logic, preferably in a form of a two-step process.
- Consider removing ability to manually set a fixed oracle price.
- Consider adding opportunity to remove unused oracle price records.

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 Rust 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

First round of testing (Sep 19th - Sep 22nd):

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

Second round of testing (Oct 14th - Oct 21st):

1. CosmWasm Smart Contracts
    (a) Repository: outposts
    (b) Commit ID: 5184e95683296559d94230943b0792bc6ad8f480
    (c) Contracts in scope:
        i. oracle
    (d) Packages in scope:
        i. outpost

Third round of testing (Nov 15th - Nov 18th):

1. CosmWasm Smart Contracts
    (a) Repository: outposts
    (b) Commit ID: f71e06b946fb754aa8ada8d4bcc1f405f6de479c

(c) Contracts in scope:

i. oracle

(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	0	1	3	1

LIKELIHOOD

IMPACT

	(HAL-01)			
(HAL-02) (HAL-03)				
		(HAL-04)		
(HAL-05)				

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) POOL PRICE SOURCE MIGHT INCREASE POTENTIAL RISKS	Medium	RISK ACCEPTED
(HAL-02) PRIVILEGED ADDRESS CAN BE TRANSFERRED WITHOUT CONFIRMATION	Low	RISK ACCEPTED
(HAL-03) FIXED PRICE SOURCE MIGHT INCREASE POTENTIAL RISKS	Low	RISK ACCEPTED
(HAL-04) ORACLE ENTRIES CANNOT BE REMOVED	Low	SOLVED - 10/04/2022
(HAL-05) OVERFLOW-CHECKS BEST PRACTICES	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) POOL PRICE SOURCE MIGHT INCREASE POTENTIAL RISKS - MEDIUM

Description:

The oracle may be using spot price source in order to provide price of a certain asset. This price is taken directly from a pool and depends on a ratio derived from the amount of each coin in the underlying pool. In case of a pool imbalance, this can be a key factor allowing for a dangerous attack. Pool imbalance happens when a significant amount of one coin is added to the pool while the other stays the same (and/or is decreased), so one asset becomes significantly more expensive in comparison to the other. This may happen as a result of a flash loan attack or a user who possesses extremely high amount of certain coins. If the pool imbalance succeeds, the oracle will display an incorrect price, which may in turn lead to price-based attacks, e.g., buying other assets at lowered prices if the oracle price at a time of pool imbalance will be used for any price calculations.

Code Location:

Below code shows the price source logic. It allows for setting up fixed, spot-supplied or tvwap based price.

Listing 1: contracts/oracle/osmosis/src/price_source.rs (Lines 46,47)

```
40 impl fmt::Display for OsmosisPriceSource {
41     fn fmt(&self, f: &mut fmt::Formatter) -> fmt::Result {
42         let label = match self {
43             OsmosisPriceSource::Fixed {
44                 price,
45             } => format!("fixed:{}", price),
46             OsmosisPriceSource::Spot {
47                 pool_id,
48             } => format!("spot:{}", pool_id),
49             OsmosisPriceSource::Twap {
50                 pool_id,
51                 window_size,
```

```
52              } => format!("twap:{}:{}", pool_id, window_size),
53          };
54          write!(f, "{}", label)
55      }
56  }
```

One of the existing unit tests have been changed in order to present the issue. First, another pool of ID 90 is created. It contains 10 times more umars tokens than pool 89.

Listing 2: contracts/oracle/osmosis/tests/helpers.rs (Line 48)

```
45      let assets = vec![coin(12345, "uosmo"), coin(888880, "umars")
↳ ];
46      deps.querier.set_query_pool_response(
47          90,
48          prepare_query_pool_response(90, &assets, &[5000u64, 5000
↳ u64], &coin(10000, "gamm/pool/90")),
49      );
```

Then a test is executed. When taking price information from an imbalanced pool (this of id 90), the target price is 10 times higher than price in pool 89.

Listing 3: contracts/oracle/osmosis/tests/test_query_price.rs (Lines 60,88)

```
32
33  #[test]
34  fn test_querying_price_spot() {
35      let mut deps = helpers::setup_test();
36
37      helpers::set_price_source(
38          deps.as_mut(),
39          "umars",
40          OsmosisPriceSource::Spot {
41              pool_id: 89,
42          },
43      );
44
45      deps.querier.set_spot_price(
```

```
46          89,
47          "umars",
48          "uosmo",
49          QuerySpotPriceResponse {
50              spot_price: Decimal::from_ratio(88888u128, 12345u128).
↳ to_string(),
51          },
52      );
53
54      let res: PriceResponse = helpers::query(
55          deps.as_ref(),
56          QueryMsg::Price {
57              denom: "umars".to_string(),
58          },
59      );
60      assert_eq!(res.price, Decimal::from_ratio(88888u128, 12345u128
↳ ));
61
62      println!("Pool 89 - Price of asset {}: {}", res.denom, res.
↳ price.to_string());
63
64      //
↳ ----------------------------------------------------------------------//
↳
65      helpers::set_price_source(
66          deps.as_mut(),
67          "umars",
68          OsmosisPriceSource::Spot {
69              pool_id: 90,
70          },
71      );
72
73      deps.querier.set_spot_price(
74          90,
75          "umars",
76          "uosmo",
77          QuerySpotPriceResponse {
78              spot_price: Decimal::from_ratio(888880u128, 12345u128)
↳ .to_string(),
79          },
80      );
81
82      let res: PriceResponse = helpers::query(
83          deps.as_ref(),
```

```
84          QueryMsg::Price {
85              denom: "umars".to_string(),
86          },
87      );
88      assert_eq!(res.price, Decimal::from_ratio(888880u128, 12345
↳ u128));
89
90      println!("Pool 90 - Price of asset {}: {}", res.denom, res.
↳ price.to_string());
91
92 }
93
94
```

It can be seen that if a pool is in an imbalance, then the price may be subject to significant fluctuations.

Risk Level:

Likelihood - 2
Impact - 5

Recommendation:

It is recommended to use more price sources at once, for example even when using the Spot price, it could be compared to current TVWAP price value in order to exclude edge cases which might indicate an attack being conducted.

Remediation plan:

RISK ACCEPTED: The Mars Protocol team accepted the risk of this finding. They will consider implementing improvements to this code in a future release.

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

Description:

An incorrect use of the update_config function from the **oracle** 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 4: contracts/oracle/base/src/contract.rs (Line 113)

```
102     fn update_config(
103         &self,
104         deps: DepsMut<C>,
105         sender_addr: Addr,
106         owner: Option<String>,
107     ) -> ContractResult<Response> {
108         let mut cfg = self.config.load(deps.storage)?;
109         if sender_addr != cfg.owner {
110             return Err(MarsError::Unauthorized {}.into());
111         };
112
113         cfg.owner = option_string_to_addr(deps.api, owner, cfg.
 ↳ owner)?;
114
115         self.config.save(deps.storage, &cfg)?;
116
117         Ok(Response::new().add_attribute("action", "outposts/
 ↳ oracle/update_config"))
118     }
```

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) FIXED PRICE SOURCE MIGHT INCREASE POTENTIAL RISKS - LOW

Description:

The oracle uses configurable price sources, one of which is a **fixed** price, which is a hardcoded value. While this might be used as a result of using another oracle (which result will be set dynamically as the fixed price), it is discouraged to use fixed prices, since they can easily become outdated and prone to potential attacks or creating unnecessary arbitrage opportunity.

Moreover, if oracle's owner account is compromised, then a potential attacker can use the fixed price setting possibility to conduct another attack on any other entity that uses prices supplied by the oracle.

Code Location:

Snippet below shows ability to set a fixed price: validate function allows several options to be set up, one of them is fixed option, which is shown in the second snippet:

Listing 5: contracts/oracle/base/src/contract.rs (Lines 132-134)

```
120     fn set_price_source(
121         &self,
122         deps: DepsMut<C>,
123         sender_addr: Addr,
124         denom: String,
125         price_source: P,
126     ) -> ContractResult<Response> {
127         let cfg = self.config.load(deps.storage)?;
128         if sender_addr != cfg.owner {
129             return Err(MarsError::Unauthorized {}.into());
130         }
131
```

```
132         price_source.validate(&deps.querier, &denom, &cfg.
    ↳ base_denom)?;
133
134         self.price_sources.save(deps.storage, denom.clone(), &
    ↳ price_source)?;
135
136         Ok(Response::new()
137             .add_attribute("action", "outposts/oracle/
    ↳ set_price_source")
138             .add_attribute("denom", denom)
139             .add_attribute("price_source", price_source.to_string
    ↳ ()))
140     }
```

Listing 6: contracts/oracle/osmosis/src/price_source.rs (Lines 71-73)

```
64 impl PriceSource<Empty> for OsmosisPriceSource {
65     fn validate(
66         &self,
67         querier: &QuerierWrapper,
68         denom: &str,
69         base_denom: &str,
70     ) -> ContractResult<()> {
71         match self {
72             OsmosisPriceSource::Fixed {
73                 ..
74             } => Ok(()),
75             OsmosisPriceSource::Spot {
76                 pool_id,
77             } => helpers::assert_osmosis_pool_assets(querier, *
    ↳ pool_id, denom, base_denom),
78             OsmosisPriceSource::Twap {
79                 pool_id,
80                 window_size,
81             } => {
82                 helpers::assert_osmosis_pool_assets(querier, *
    ↳ pool_id, denom, base_denom)?;
83
84                 if *window_size > TWO_DAYS_IN_SECONDS {
85                     Err(ContractError::InvalidPriceSource {
86                         reason: format!(
87                             "expecting window size to be within {}
    ↳  sec",
88                             TWO_DAYS_IN_SECONDS
```

```
89                          ),
90                      })
91                  } else {
92                      Ok(())
93                  }
94              }
95              OsmosisPriceSource::LiquidityToken {
96                  ..
97              } => Ok(()),
98          }
99      }
```

Risk Level:

Likelihood - 1
Impact - 4

Recommendation:

Consider removing ability to set a fixed price of an asset.

Remediation plan:

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

3.4 (HAL-04) ORACLE ENTRIES CANNOT BE REMOVED - LOW

Description:

There is no possibility to remove entries that have already been saved into the oracle. In case an error occurs, or a delisting is required, there will be no possibility to remove an asset from an oracle. Consequently, any party that is still relying on the oracle may keep receiving data on an asset on which it should not. Also, over time the storage might grow in an uncontrolled way, thus the ability to clear some entries may be useful.

Code Location:

Listing 7: contracts/oracle/base/src/contract.rs (Lines 75-78)

```
65     pub fn execute(
66         &self,
67         deps: DepsMut<C>,
68         info: MessageInfo,
69         msg: ExecuteMsg<P>,
70     ) -> ContractResult<Response> {
71         match msg {
72             ExecuteMsg::UpdateConfig {
73                 owner,
74             } => self.update_config(deps, info.sender, owner),
75             ExecuteMsg::SetPriceSource {
76                 denom,
77                 price_source,
78             } => self.set_price_source(deps, info.sender, denom,
↳ price_source),
79         }
80     }
```

Risk Level:

Likelihood - 3
Impact - 2

Recommendation:

Consider adding remove oracle price utility.

Remediation plan:

SOLVED: The issue was solved in commit 5184e95683296559d94230943b0792bc6ad8f480.

3.5 (HAL-05) OVERFLOW-CHECKS BEST PRACTICES - INFORMATIONAL

Description:

The `overflow-checks` setting of the `Cargo.toml` controls the `-C overflow-checks` flag, which controls the behavior of runtime integer overflow. When overflow-checks are enabled, a panic will occur on overflow, thus protecting the contracts against that attack vector.

It was found that the `overflow-checks` is enforced at the workspace level, which covers the whole nested contracts. However, in a security-in-depth approach, it is recommended as a best practice to enable that check on each contract in case someone refactors the project and removes `[profile.release]` in the workspace's `Cargo.toml`, or reuse the contract in a different project that doesn't have the check.

Code Location:

Listing 8: Resources affected

```
contracts/oracle/base/Cargo.toml
contracts/oracle/osmosis/Cargo.toml
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended explicitly to enable overflow checks on each individual contract and package. That measure helps when the project is refactored to avoid unintended consequences.

Remediation plan:

ACKNOWLEDGED: The Mars Protocol team acknowledged this finding.

THANK YOU FOR CHOOSING

// HALBORN