// HALBORN

# Sienna.Network Lending Protocol

## CosmWasm Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **February 15th, 2022 - March 4th, 2022**

Visit: **Halborn.com**

# DOCUMENT REVISION HISTORY

| VERSION | MODIFICATION | DATE | AUTHOR |
|---|---|---|---|
| 0.1 | Document Creation | 02/15/2022 | Alexis Fabre |
| 0.2 | Document Updates | 03/04/2022 | Alexis Fabre |
| 0.3 | Draft Version | 03/08/2022 | Alexis Fabre |
| 0.4 | Draft Review | 03/11/2022 | Gabi Urrutia |
| 1.0 | Remediation Plan | 03/30/2022 | Michal Bazyli |
| 1.1 | Remediation Plan Review | 03/30/2022 | 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 |
| Alexis Fabre | Halborn | Alexis.Fabre@halborn.com |

# EXECUTIVE OVERVIEW

## 1.1 INTRODUCTION

engaged Halborn to conduct a security audit on their smart contracts beginning on February 15th, 2022 and ending on March 4th, 2022 . The security assessment was scoped to the smart contracts provided to the Halborn team.

## 1.2 AUDIT SUMMARY

The team at Halborn was provided two weeks 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 improvements to reduce the likelihood and impact of risks, which were partially addressed by Sienna.Network team. The main ones are the following:

- Ensure that a user can enter the same market only once.
- Ensure that a low asset price do not get truncated to zero when calculating liquidity.

External threats, such as financial related attacks, oracle attacks, and inter-contract functions and calls should be validated for expected logic and state.

# 1.3 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

## 1.4 SCOPE

1. CosmWasm Smart Contracts
   (a) Repository: contracts-lend
   (b) Commit ID: dbe3c8688e75dd0b89634e6f22f861e44c849f06
   (c) Contracts in scope:
      i. interest_model
      ii. market
      iii. mock_band_oracle
      iv. oracle
      v. overseer

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

# 2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

| CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL |
|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 5 |

LIKELIHOOD

IMPACT

| | | | | |
|---|---|---|---|---|
| | | | | (HAL-01) |
| | (HAL-02) | | | |
| (HAL-03) | | | | |
| (HAL-04)<br>(HAL-05)<br>(HAL-06) | | | | |
| (HAL-07)<br>(HAL-08) | | | | |

# EXECUTIVE OVERVIEW

| SECURITY ANALYSIS | RISK LEVEL | REMEDIATION DATE |
|---|---|---|
| (HAL-01) USERS CAN ENTER MULTIPLE TIMES THE SAME MARKET | Critical | SOLVED - 03/07/2022 |
| (HAL-02) TRUNCATED DECIMALS CAN LEAD TO UNLIMITED LOANS | Medium | SOLVED - 03/30/2022 |
| (HAL-03) NEW BORROWING PARAMETERS CAN BE APPLIED TO ONGOING LOANS | Low | RISK ACCEPTED |
| (HAL-04) ADMIN CAN SET WORTHLESS TOKENS AS COLLATERAL | Informational | ACKNOWLEDGED |
| (HAL-05) ORACLE RISK ALLOWING FREE LOANS | Informational | ACKNOWLEDGED |
| (HAL-06) MISSING SANITY CHECK ON MARKET CONFIGURATION | Informational | SOLVED - 03/30/2022 |
| (HAL-07) OVERSEER CANNOT DE-LIST MARKETS | Informational | ACKNOWLEDGED |
| (HAL-08) WRONG LOGGED DATA | Informational | SOLVED - 03/30/2022 |

# FINDINGS & TECH DETAILS

## 3.1 (HAL-01) USERS CAN ENTER MULTIPLE TIMES THE SAME MARKET - CRITICAL

Description:

Sienna.Network Lending protocol allows users to supply and borrow from "markets", represented by a governance whitelisted underlying token (eg. sATOM, sSCRT. . . ).

In order to borrow from one market, a user must follow the following steps:

- Alice sends snip20 tokens (eg. 10 sATOM) to the overseer, triggering the deposit function.
- Then, she calls the enter function, that will signal the overseer to consider the deposit as collateral. Suppose that 1 sATOM is worth 30$ with an LTV of 50%, Alice will be able to borrow 150$ worth of other tokens.

It was possible for Alice to call the function enter multiple times, mutliplying the collateral value of her deposit:

- Alice sends snip20 tokens (eg. 10 sATOM, collateral value of 150$ as previously calculated) to the overseer, triggering the deposit function.
- Then, she calls the enter function three times. The overseer will add three times the collateral value of the deposit, and Alice will be able to borrow $(3 * 150\$)$ 450$ worth of tokens, based on an initial 300$ deposit.

She can repay her debt and/or borrow even more on the protocol, until she drained all the liquidity of the protocol. **All provided funds by legitimate users will be lost**.

A proof of concept is described here.

Code Location:

The first listing represents the vulnerability entry point, where the attackers can provide multiple time the same market address. The overseer will transmit the markets into add_markets:

**Listing 1: contracts/lend/overseer/src/state.rs (Line 238)**

```
231     #[handle]
232     fn enter(markets: Vec<HumanAddr>) -> StdResult<HandleResponse>
 ↳  {
233         let account = Account::new(&deps.api, &env.message.sender)
 ↳ ?;
234         let ids = markets.iter()
235             .map(|x| Markets::get_id(deps, x))
236             .collect::<StdResult<Vec<u64>>>()?;
237
238         account.add_markets(&mut deps.storage, ids)?;
239         Ok(HandleResponse {
240             messages: vec![],
241             log: vec![log("action", "enter")],
242             data: None,
243         })
244     }
```

Next, the add_market function only checks that the user enter less than a maximum number of markets. After that, the function will push all markets from the arguments to the storage, even if some id are already stored.

**Listing 2: contracts/lend/overseer/src/contract.rs (Lines 209-211)**

```
195 pub fn add_markets<S: Storage>(
196     &self,
197     storage: &mut S,
198     ids: Vec<u64>
199 ) -> StdResult<()> {
200     let mut markets = self.load_markets(storage)?;
201
202     if markets.len() + ids.len() > MAX_MARKETS_ENTERED {
203         return Err(StdError::generic_err(format!(
```

```
204                 "Cannot enter more than {} markets at a time.",
205                 MAX_MARKETS_ENTERED
206             )));
207         }
208
209         for id in ids {
210             markets.push(id);
211         }
212
213         self.save_markets(storage, &markets)
214     }
```

The remove_market function do not properly removes all markets that have the same id from the storage:

**Listing 3: contracts/lend/overseer/src/state.rs (Lines 259-260)**

```
252 pub fn remove_market<S: Storage>(
253     &self,
254     storage: &mut S,
255     id: u64
256 ) -> StdResult<()> {
257     let mut markets = self.load_markets(storage)?;
258     if let Some(index) = markets.iter().position(|x| *x == id) {
259         markets.swap_remove(index);
260         self.save_markets(storage, &markets)
261     } else {
262         Ok(())
263     }
264 }
```

Risk Level:

**Likelihood - 5**

**Impact - 5**

Recommendation:

Make sure that it is not possible to enter a market multiple times. Additionally, fix `remove_market` to delete all items that match the provided `id`.

Remediation Plan:

**SOLVED:** The issue was fixed in commit `557f5c92d5d9ecc9d7cdee9318aa69bd842a8c31`.

## 3.2 (HAL-02) TRUNCATED DECIMALS CAN LEAD TO UNLIMITED LOANS - MEDIUM

Description:

The oracle uses 128 bits unsigned integers to store the price, where the 18 first digits represents the decimals, and the overseer converts that into 256 bits decimals with 18 digits to perform arithmetic operations when calculating liquidity and collateral values in the `calculate_liquidity` function.

In some cases where the product of the borrow amount and the price of an asset is very low is lower than `1.0`, the result is truncated by the conversion to an integer, resulting in a free loan.

For example, if a token A is worth 0.1 USD and has 6 digits and Alice borrows 9 units of token A, the product of the two equals 0.9 units.USD and is lower than one. Therefore, Alice borrows 9 units of token A freely.

Note that the vulnerability is mitigated when the price of the asset is greater than one.

The impact is also greatly reduced by the amount of decimals of a token. If token A has 6 decimals, Alice would have borrowed 0.000009 token A at the price of 0.1 USD, which represents 0.0000009 USD and the action is not profitable because of transaction fees.

Code Location:

**Listing 4: contracts/lend/overseer/src/contract.rs (Line 531)**

```
514 let is_target_asset = target_asset == market.contract.address;
515 let snapshot = query_account(&deps.querier, market.contract,
 ↳ method.clone(), block)?;
516 let price = query_price(
517     &deps.querier,
```

```
518     oracle.clone(),
519     market.symbol.into(),
520     QUOTE_SYMBOL.into(),
521     None,
522 )?;
523 let conversion_factor = ((market.ltv_ratio * snapshot.
 ↳ exchange_rate)? * price.rate)?;
524 total_collateral = (Uint256::from(snapshot.sl_token_balance)
525     .decimal_mul(conversion_factor)?
526     + total_collateral)?;
527 total_borrowed =
528     (Uint256::from(snapshot.borrow_balance).decimal_mul(price.rate
 ↳ )? + total_borrowed)?;
529 if is_target_asset {
530     total_borrowed = (redeem_amount.decimal_mul(conversion_factor)
 ↳ ? + total_borrowed)?;
531     total_borrowed = (borrow_amount.decimal_mul(price.rate)? +
 ↳ total_borrowed)?;
532 }
```

Risk Level:

**Likelihood - 2**
**Impact - 4**

Recommendation:

It is advised to perform arithmetic on integers to avoid decimal approximations. As an alternate mitigation, a verification can be performed on `total_collateral` and `total_borrowed` to revert the transaction when one or both is equal to zero.

Remediation plan:

**SOLVED:** The issue was fixed in commit `6acd2bcc9966bc671bd60929d7b72393838fb708`.

## 3.3 (HAL-03) NEW BORROWING PARAMETERS CAN BE APPLIED TO ONGOING LOANS - LOW

Description:

The change_market function in **contracts/lend/overseer/src/contract.rs** allows the unrestricted modification of ltv_ratio, affecting how much borrowing power a specific token yields. The change_config function of the same contract also allows the modification of close_factor, that defines the liquidation threshold. 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 the likelihood for this to happen is low because the **overseer** contract is intended to be owned by governance indefinitely, who is the responsible one for this operation.

Code Location:

Here, the administrator can update a market ltv_ratio. The validate function verifies if the value of ltv_ratio is lesser or equal than 1 and also greater than 0. However, it does not verify if the new value has a significant difference with the previous one.

**Listing 5: contracts/lend/overseer/src/contract.rs (Lines 297-304)**

```
294 #[handle]
295 #[require_admin]
296     Markets::update(deps, &market, |mut m| {
297         m.ltv_ratio = ltv_ratio;
298         m.validate()?;
299         m.symbol = symbol.clone();
300         Ok(m)
301     })?;
302     let messages = if update_oracle {
303         let oracle = Contracts::load_oracle(deps)?;
```

```
304         vec![CosmosMsg::Wasm(WasmMsg::Execute {
305             contract_addr: oracle.address,
306             callback_code_hash: oracle.code_hash,
307             send: vec![],
308             msg: to_binary(&OracleHandleMsg::UpdateAssets {
309                 assets: vec![Asset {
310                     address: stored_market.contract.address,
311                     symbol,
312                 }],
313             })?,
314         })]
315     } else {
316         vec![]
317     };
318     Ok(HandleResponse {
319         messages,
320         log: vec![],
321         data: None
322     })
323 }
```

Here the admin can update the close_factor without any validation:

**Listing 6: contracts/lend/overseer/src/contract.rs (Lines 341-343)**

```
331 #[handle]
332 #[require_admin]
333 fn change_config(
334     premium_rate: Option<Decimal256>,
335     close_factor: Option<Decimal256>
336 ) -> StdResult<HandleResponse> {
337     let mut constants = Constants::load(&deps.storage)?;
338     if let Some(premium_rate) = premium_rate {
339         constants.set_premium(premium_rate)?;
340     }
341     if let Some(close_factor) = close_factor {
342         constants.set_close_factor(close_factor)?;
343     }
344     Constants::save(&mut deps.storage, &constants)?;
345     Ok(HandleResponse {
346         messages: vec![],
347         log: vec![
348             log("action", "change_config"),
```

```
            log("premium_rate", constants.premium()),
            log("close_factor", constants.close_factor())
        ],
        data: None
    })
}
```

Risk Level:

**Likelihood - 1**
**Impact - 3**

Recommendation:

Update the logic of change_market and change_config functions to include a **ramp change schema** for close_factor and ltv_ratio 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 Sienna.Network accepted the risk of this finding.

## 3.4 (HAL-04) ADMIN CAN SET WORTHLESS TOKENS AS COLLATERAL - INFORMATIONAL

Description:

In the overseer contract, the change_market function allows an administrator to update the LTV ratio and/or the symbol of a designated market (e.g. token whitelisted in the lending platform). In order to restrict worthless tokens to be used as collateral, the function controls that the token price is not zero by querying the oracle: query_price(SYMBOL, QUOTE). However, when updating the SYMBOL at the same time as the LTV value, the aforementioned control is performed on the new asset SYMBOL.

That logic flaw allows an administrator to:
- First call: Set the SYMBOL of the market to something that has a non-null price (eg. SCRT)
- Second call: Set the LTV value of the market with the wanted value
- Third call: Set the SYMBOL of the market back to the previous one

That will result in a worthless token able to count as collateral. This couldn't be escalated to a bigger finding.

Code Location:

**Listing 7: contracts/lend/overseer/src/contract.rs (Lines 274,280,285-290)**

```
264 #[handle]
265 #[require_admin]
266 fn change_market(
267     market: HumanAddr,
268     ltv_ratio: Option<Decimal256>,
269     symbol: Option<String>,
270 ) -> StdResult<HandleResponse> {
271     let (_, stored_market) = Markets::get_by_addr(deps, &market)?;
272
```

```
273     let update_oracle = symbol.is_some();
274     let symbol = symbol.unwrap_or(stored_market.symbol);
275
276     let ltv_ratio = if let Some(ltv_ratio) = ltv_ratio {
277         let price = query_price(
278             &deps.querier,
279             Contracts::load_oracle(deps)?,
280             symbol.clone().into(),
281             QUOTE_SYMBOL.into(),
282             None,
283         )?;
284
285         // Can't set collateral factor if the price is 0
286         if price.rate == Decimal256::zero() {
287             return Err(StdError::generic_err(
288                 "Cannot set LTV ratio if the price is 0",
289             ));
290         }
291
292         ltv_ratio
293     } else {
294         stored_market.ltv_ratio
295     };
```

Risk Level:

**Likelihood - 1**
**Impact - 2**

Recommendation:

It is recommended to either split the function into two specialized functions: `change_market_symbol` and `change_market_ltv`, or update the logic to remove this possibility.

Remediation plan:

**ACKNOWLEDGED:** The Sienna.Network acknowledged this finding.

## 3.5 (HAL-05) ORACLE RISK ALLOWING FREE LOANS - INFORMATIONAL

Description:

The overseer relies on an external contract, the Band oracle, to get the prices of underlying assets. That way, the overseer can calculate the collateral value of each asset. It can happen that oracles get attacked, resulting in significant losses. Sienna.Network lending protocol would in that case face unlimited borrow on an asset with a value of 0 until the market is empty.

The probability of such an event is very low because the Band protocol uses a volume-weighted average price, aggregating prices from multiple sources, which greatly reduces the risk of hacks.

Code Location:

**Listing 8: contracts/lend/overseer/src/contract.rs (Lines 523-529)**

```
513 for market in markets {
514     let is_target_asset = target_asset == market.contract.address;
515     let snapshot = query_account(&deps.querier, market.contract,
 ↳ method.clone(), block)?;
516     let price = query_price(
517         &deps.querier,
518         oracle.clone(),
519         market.symbol.into(),
520         QUOTE_SYMBOL.into(),
521         None,
522     )?;
523     let conversion_factor = ((market.ltv_ratio * snapshot.
 ↳ exchange_rate)? * price.rate)?;
524     total_collateral = (Uint256::from(snapshot.sl_token_balance)
525         .decimal_mul(conversion_factor)?
526         + total_collateral)?;
527     total_borrowed =
528         (Uint256::from(snapshot.borrow_balance).decimal_mul(price.
 ↳ rate)? + total_borrowed)?;
```

```
529     if is_target_asset {
530         total_borrowed = (redeem_amount.decimal_mul(
 ↳ conversion_factor)? + total_borrowed)?;
531         total_borrowed = (borrow_amount.decimal_mul(price.rate)? +
 ↳  total_borrowed)?;
532     }
533 }
```

Risk Level:

**Likelihood - 1**
**Impact - 2**

Recommendation:

It is advised to implement a detection mechanism on the price variation, for example by comparing priceDifference / time between two price queries at different time, with a threshold (ie. 10%), so that the overseer can react accordingly (freezing the contract, changing oracle, etc. . . ).

Remediation plan:

**ACKNOWLEDGED:** The Sienna.Network acknowledged this finding. They also stated that the volume weighted average price of the Band protocol should be safe enough.

## 3.6 (HAL-06) MISSING SANITY CHECK ON MARKET CONFIGURATION - INFORMATIONAL

Description:

At market contract `init` and `update_config`, the configuration is stored without any verification. For example, the reserve factor is expected to be lower than 1 but is not validated. That can lead to errors when trying to liquidate positions until the parameter is eventually corrected.

Code Location:

**Listing 9: contracts/lend/market/src/contract.rs**

```
353 #[handle]
354 #[require_admin]
355 fn update_config(
356     interest_model: Option<ContractLink<HumanAddr>>,
357     reserve_factor: Option<Decimal256>,
358     borrow_cap: Option<Uint256>,
359 ) -> StdResult<HandleResponse> {
360     let mut config = Constants::load_config(&deps.storage)?;
361     if let Some(interest_model) = interest_model {
362         Contracts::save_interest_model(deps, &interest_model)?;
363     }
364
365     if let Some(reserve_factor) = reserve_factor {
366         config.reserve_factor = reserve_factor;
367         Constants::save_config(&mut deps.storage, &config)?;
368     }
369
370     if let Some(borrow_cap) = borrow_cap {
371         Global::save_borrow_cap(&mut deps.storage, &borrow_cap)?;
372     }
373
374     Ok(HandleResponse::default())
375 }
```

Risk Level:

**Likelihood - 1**
**Impact - 2**

Recommendation:

It is advised to validate new configuration parameters to protect the contract of unwanted behaviour.

Remediation plan:

**SOLVED:** The issue was fixed in commit 74e01376630f46cd606d16c4c92decbe49d4ee83.

## 3.7 (HAL-07) OVERSEER CANNOT DE-LIST MARKETS - INFORMATIONAL

Description:

The overseer allows an admin to whitelist and register a market on which users can supply and borrow the underlying asset. In case the underlying asset is deemed malicious after being whitelisted, the only solution is to use the market contract's kill-switch.

Note that the front-end can choose to hide blacklisted markets, even if they are still valid in the blockchain.

Code Location:

Whitelisting structure do not allow the protocol to save a market with **save_market_contract** but not to remove a listed market.

**Listing 10: contracts/lend/overseer/src/state.rs**

```
65 impl Whitelisting {
66     const KEY_MARKET_CONTRACT: &'static [u8] = b"market_contract";
67     const KEY_PENDING: &'static [u8] = b"pending";
68     #[inline]
69     pub fn save_market_contract(
70         storage: &mut impl Storage,
71         contract: &ContractInstantiationInfo
72     ) -> StdResult<()> {
73         save(storage, Self::KEY_MARKET_CONTRACT, contract)
74     }
75
76     #[inline]
77     pub fn load_market_contract(
78         storage: &impl Storage
79     ) -> StdResult<ContractInstantiationInfo> {
80         Ok(load(storage, Self::KEY_MARKET_CONTRACT)?.unwrap())
81     }
82     pub fn set_pending(
83         storage: &mut impl Storage,
84         market: &Market<HumanAddr>
```

```
85      ) -> StdResult<()> {
86          save(storage, Self::KEY_PENDING, market)
87      }
88      pub fn pop_pending(
89          storage: &mut impl Storage
90      ) -> StdResult<Market<HumanAddr>> {
91          let result: Option<Market<HumanAddr>> =
92              load(storage, Self::KEY_PENDING)?;
93          match result {
94              Some(market) => {
95                  storage.remove(Self::KEY_PENDING);
96                  Ok(market)
97              },
98              None => Err(StdError::unauthorized())
99          }
100     }
101 }
```

Risk Level:

**Likelihood - 1**
**Impact - 1**

Recommendation:

It is advised to add a remove_market to the overseer contract

Remediation plan:

**ACKNOWLEDGED:** The Sienna.Network acknowledged this finding.

## 3.8 (HAL-08) WRONG LOGGED DATA - INFORMATIONAL

Description:

In the overseer contract, the function register_oracle returns the log register_interest_token in the response message. That could bring confusion in the deployment of the contract.

Code Location:

**Listing 11: contracts/lend/overseer/src/contract.rs (Lines 95,108)**

```
94 #[handle]
95 fn register_oracle() -> StdResult<HandleResponse> {
96     let mut oracle = Contracts::load_oracle(deps)?;
97
98     if oracle.address != HumanAddr::default() {
99         return Err(StdError::unauthorized());
100    }
101
102    oracle.address = env.message.sender;
103    Contracts::save_oracle(deps, &oracle)?;
104
105    Ok(HandleResponse {
106        messages: vec![],
107        log: vec![
108            log("action", "register_interest_token"),
109            log("oracle_address", oracle.address),
110        ],
111        data: None,
112    })
113 }
```

Risk Level:

**Likelihood - 1**

**Impact - 1**

Recommendation:

Change the log to `register_oracle` instead.

Remediation plan:

**SOLVED:** The issue was fixed in commit `406be132d4f5c8a29670afcc9ca5c1a2473b7cf3`.

# 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.

| ID | package | Short Description |
|---|---|---|
| RUSTSEC-2020-0159 | chrono | Potential segfault in ‘localtime_r‘ |
| RUSTSEC-2021-0076 | libsecp256k1 | libsecp256k1 allows overflowing signatures |
| RUSTSEC-2020-0071 | time | Potential segfault in the time crate |

THANK YOU FOR CHOOSING

// HALBORN