// HALBORN

Nolus - Money Market

CosmWasm Smart Contract Security Assessment

Prepared by: **Halborn**

Date of Engagement: **May 22nd, 2023 - July 7th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	06/05/2023	Alexis Fabre
0.2	Document Update	06/05/2023	Alexis Fabre
0.3	Document Update	07/03/2023	Elena Maranon
0.4	Draft Review	07/10/2023	Alp Onaran
0.5	Draft Review	07/10/2023	Piotr Cielas
0.6	Draft Review	07/11/2023	Gabi Urrutia
1.0	Remediation Plan	08/17/2023	Elena Maranon
1.1	Remediation Plan Review	08/17/2023	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
Alexis Fabre	Halborn	alexis.fabre@halborn.com
Elena Maranon	Halborn	elena.maranon@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Nolus is a DeFi leasing platform built on the Cosmos SDK that introduces a peer-to-peer marketplace where lenders can provide liquidity and borrowers can lease these tokens to execute financial transactions. It employs a collateral-based system where borrowers must provide collateral to secure their lease. This collateral, in the event of a default, is used to repay the lenders.

One of Nolus key features is the fixed borrower terms of interest throughout the lease contract. This aspect provides a reliable and predictable outlook for future cash flows and reward distributions for lenders. Furthermore, the platform cumulative profits, which are gathered from the protocol lease contracts, swap spreads, and transaction fees, are employed to repurchase NLS tokens from the open market, driving price appreciation of the token.

The platform uses several components to ensure optimal operation, such as Market Data Price Oracle and Global Time Oracle, which provide accurate market data and time-based updates respectively. The pricing data is further enhanced by a Price Feeder optimizing security and recent price observation.

In terms of risk management, Nolus has implemented liquidation mechanisms to manage lease defaults. If a borrower's liability exceeds a certain threshold, or they fail to pay their interest, a liquidation process is triggered. The system sells the borrower collateral, and the proceeds are used to repay the lenders.

Nolus engaged Halborn to conduct a security assessment on their smart contracts beginning on May 22nd, 2023 and ending on July 7th, 2023. The security assessment was scoped to the smart contracts provided in the Nolus repository. Commit hashes and further details can be found in the Scope section of this report.

1.2 ASSESSMENT SUMMARY

The team at Halborn was provided seven weeks for the engagement and assigned a full-time security engineer to verify 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 assessment 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, that were mostly addressed by Nolus. The main ones were the following:

- Enforce a slippage limit for the Swap transactions.
- Improve the resiliency of ACK delivery messages.

1.3 SCOPE

Code repository: nolus-money-market

1. CosmWasm LP Strategy Smart Contract
   (a) Commit ID: 430820f
   (b) Contract in scope:
      i. admin
      ii. dispatcher
      iii. lease
      iv. leaser
      v. lpp
      vi. oracle

vii. profit
viii. timealarms
ix. treasury

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

1.4 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 assessment. 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 assessment:

- Research into the architecture, purpose, and use of the platform.
- Smart contract manual code review and walkthrough to identify any logic issue.
- Thorough assessment of safety and usage of critical Rust variables and functions in scope that could lead to arithmetic vulnerabilities.
- Scanning of Rust files for common vulnerabilities (cargo audit).
- Local deployment (Nolus node, Osmosis testnet and Hermes-Relayer).

2. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

2.1 EXPLOITABILITY

Attack Origin (AO):

Captures whether the attack requires compromising a specific account.

Attack Cost (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

Attack Complexity (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

Metrics:

Exploitability Metric (m_E)	Metric Value	Numerical Value
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

2.2 IMPACT

Confidentiality (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

Integrity (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

Availability (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

Deposit (D):

Measures the impact to the deposits made to the contract by either users or owners.

Yield (Y):

Measures the impact to the yield generated by the contract for either users or owners.

Metrics:

Impact Metric (m_I)	Metric Value	Numerical Value
Confidentiality (C)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Integrity (I)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Availability (A)	None (A:N)	0
	Low (A:L)	0.25
	Medium (A:M)	0.5
	High (A:H)	0.75
	Critical	1
Deposit (D)	None (D:N)	0
	Low (D:L)	0.25
	Medium (D:M)	0.5
	High (D:H)	0.75
	Critical (D:C)	1
Yield (Y)	None (Y:N)	0
	Low (Y:L)	0.25
	Medium: (Y:M)	0.5
	High: (Y:H)	0.75
	Critical (Y:H)	1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

2.3 SEVERITY COEFFICIENT

Reversibility (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

Scope (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

Coefficient (C)	Coefficient Value	Numerical Value
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

Severity	Score Value Range
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	1	6

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) NO SLIPPAGE PROTECTION	Medium (5.0)	ACKNOWLEDGED
(HAL-02) THE LEASE CONTRACT COULD BE BLOCKED DURING STATE TRANSITION	Low (2.8)	SOLVED - 08/17/2023
(HAL-03) TVL RATES CANNOT BE UPDATED	Informational (1.0)	SOLVED - 08/17/2023
(HAL-04) ICA TRANSACTIONS DO NOT TIP THE RECEIVER RELAYER	Informational (0.0)	ACKNOWLEDGED
(HAL-05) REDUNDANT FIELD IN THE LOAN STRUCT	Informational (0.0)	SOLVED - 08/17/2023
(HAL-06) DUPLICATE STORAGE READ	Informational (0.0)	SOLVED - 08/17/2023
(HAL-07) OUTDATED OBSERVATIONS CAN BE STORED	Informational (0.0)	SOLVED - 08/17/2023
(HAL-08) RELAYER TIPS ARE HARDCODED	Informational (0.0)	ACKNOWLEDGED

FINDINGS & TECH DETAILS

4.1 (HAL-01) NO SLIPPAGE PROTECTION - MEDIUM (5.0)

Description:

The exact_amount_in function in the packages/swap/src/trx.rs file does not consider slippage. This means that the user may not receive the exact amount of tokens that they expect.

In large swap transactions, there is a risk known as 'sandwich attacks'. These occur when MEV algorithms on Osmosis purchase significant amounts of the asset involved in the swap. This action drives the price of the asset up before the swap occurs. These algorithms then sell the asset at this higher price, making a profit. As a result, the user conducting the swap might end up paying more for the asset than they initially expected.

Code Location:

- packages/swap/src/trx.rs#L24-L51

Listing 1: nolus-money-market/packages/swap/src/trx.rs (Lines 38,41,46)

```
24 pub fn exact_amount_in<G>(
25     trx: &mut Transaction,
26     sender: HostAccount,
27     token_in: &CoinDTO<G>,
28     swap_path: &SwapPath,
29 ) -> Result<()>
30 where
31     G: Group,
32 {
33     // TODO bring the token balances, weights and swapFee-s from
↳ the DEX pools
34     // into the oracle in order to calculate the tokenOut as per
↳ the formula at
35     // https://docs.osmosis.zone/osmosis-core/modules/gamm/#swap.
36     // Then apply the parameterized maximum slippage to get the
↳ minimum amount.
```

```
37      // For the first version, we accept whatever price impact and
↳ slippage.
38      const MIN_OUT_AMOUNT: &str = "1";
39      let routes = to_route(swap_path)?;
40      let token_in = Some(to_cwcoin(token_in)?);
41      let token_out_min_amount = MIN_OUT_AMOUNT.into();
42      let msg = MsgSwapExactAmountIn {
43          sender: sender.into(),
44          routes,
45          token_in: token_in.map(Into::into),
46          token_out_min_amount,
47      };
48
49      trx.add_message(REQUEST_MSG_TYPE, msg);
50      Ok(())
51  }
```

BVSS:

AO:A/AC:L/AX:M/C:N/I:N/A:N/D:H/Y:N/R:N/S:U (5.0)

Recommendation:

It is recommended to consider slippage when calculating the minimum amount of tokens that the user should receive. Note that the TODO comment in the code snippet above indicates that this is already planned.

Remediation plan:

ACKNOWLEDGED: The Nolus team acknowledged this finding.

4.2 (HAL-02) THE LEASE CONTRACT COULD BE BLOCKED DURING STATE TRANSITION - LOW (2.8)

Description:

The Leaser creates a new Lease contract each time a user makes a open_lease request. This contract works like a state machine, moving from one state to another depending on the status of the loan granted to the user. There are 10 different states, divided into four main phases (opening, opened, paid, closed). In each phase, the transition from one state to another is done automatically, without any external interaction.

Since this money market is going to operate with different currencies, both the communication with a DEX and the use of an Oracle are essential. The DEX used currently is Osmosis, which involves IBC communication for the exchange of currencies. IBC communication implies that transactions are not atomic, so synchronization and resilience of the different components in the communication is critical. The Neutron SDK is used as middleware to manage IBC packets during communications.

It has been observed during the dynamic tests that, in case of error in the Lease contract, for example, if the oracle price feed does not work correctly, the error at the moment of trying to buy the asset is not propagated and handled by the contract, it is only notified to the ContractManager module, so the Lease contract has no information to revert the operations due to the error, nor the possibility to advance to the next state, leaving the contract in a state of permanent blocking.

BVSS:

AO:S/AC:L/AX:L/C:N/I:N/A:C/D:H/Y:H/R:N/S:U (2.8)

Recommendation:

It is recommended to ensure that all possible failures in non-atomic transactions, especially those related to interchain communication problems, are handled by the contract, reverting to a previous state if necessary.

Remediation Plan:

SOLVED: The Nolus team has addressed the issues inherited from the use of NeutronSDK. These problems could be grouped into four categories:

- The handling of 'sudo' messages is not atomic: The handling of sudo messages is not atomic: A failure in the processing of a sudo submessage does not revert the state changes made by the processing of the main sudo message. The Neutron team has solved this with a new version.
- IBC messages sent as part of the handling of a 'sudo' message are not forwarded: On successful processing of a ‘sudo’ message, the Neutron SDK ignores the emitted events. The Neutron team has fixed this with a new version.
- 'Out-of-gas' is treated as an error returned by the smart contract : These errors are received and logged in its contractmanager module. Instead, out-of-gas errors should be returned to the relayers reversing the transactions and allowing the next IBC packet delivery attempts.
  The Neutron team plans to fix this in their next release, so as a workaround until the solution is ready, the Nolus team has developed and deployed a lease ‘healing’ feature that recovers leases left in an invalid state.
  Commit IDs: 78734bf, 9b18a47
- Guaranteed delivery of Dex ACKs: According to a NeutronSDK design decision, errors returned by smart contracts when delivering acks/timeouts/errors are received and logged in their contractmanager module. For now, there is no means for contracts to request a subsequent delivery.

Leveraging Nolus Money Market's time alarms, Nolus team has developed a solution to guarantee the delivery of Dex ACKs. It is based on the Cosmwasm actor model and the ability to gain control of errors occurring in a sub-message. If the first delivery fails, the `ResponseDelivery` leverages the guaranteed delivery of time alarms by scheduling a time alarm to make a delivery attempt on the next alarm sending cycle.

Commit IDs: 16b1886, 4e84944

4.3 (HAL-03) TVL RATES CANNOT BE UPDATED - INFORMATIONAL (1.0)

Description:

The configuration of TVL-to-APR rates in the **Rewards Dispatcher** contract is performed on contract instantiation. Although the contract allows updating some configuration parameters by a governance proposal, the only value that can be updated is the cadence_hours. The RewardScale parameter tvl_to_apr cannot be updated after instantiation.

Code Location:

- contracts/dispatcher/src/contract.rs#L100-L106

Listing 2: nolus-money-market/contracts/dispatcher/src/contract.rs (Line 103)

```
100 pub fn sudo(deps: DepsMut<'_>, _env: Env, msg: SudoMsg) ->
 ↳ ContractResult<CwResponse> {
101     match msg {
102         SudoMsg::Config { cadence_hours } => {
103             Config::update(deps.storage, cadence_hours).map(|()|
 ↳ response::empty_response())
104         }
105     }
106 }
107
```

BVSS:

AO:S/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:M/R:N/S:U (1.0)

Recommendation:

It is recommended to consider the possibility of modifying the TVL-to-APR parameter in case it needs some modification in the future due to strategic decisions.

Remediation Plan:

SOLVED: The Nolus team has solved this issue in the commit ID 65aff6ab19256578271cbaba70c4519313d24497.

4.4 (HAL-04) ICA TRANSACTIONS DO NOT TIP THE RECEIVER RELAYER - INFORMATIONAL (0.0)

Description:

Interchain Standard 29 (ICS 29), also known as “relayer fees”, is a feature aimed at incentivizing Inter-Blockchain Communication (IBC) infrastructure within the Cosmos ecosystem. ICS 29 is designed to make the job of relayers more sustainable by providing an incentivization layer that rewards relayer operators for successfully relaying packets. The standard defines three types of relayers: Forward relayer, who submits the recv_packet message for a given packet; Reverse relayer, who submits the ack_packet message for a given packet; and Timeout relayer, who submits the timeout_packet message for a given packet

It was identified that the ICA (Interchain Accounts) transactions do not tip the receiver relayer. Not tipping for the reception of packets (recv _fee) in the IBC protocol may lead to a reduced incentive for relayers to perform this operation, potentially impacting the reliability and efficiency of packet transmission across the blockchain networks.

Code Location:

- packages/platform/src/ica.rs#L61-L89

Listing 3

```
pub fn submit_transaction<Conn, M, C>(
    connection: Conn,
    trx: Transaction,
    memo: M,
    timeout: Duration,
    ack_tip: Coin<C>,
    timeout_tip: Coin<C>,
) -> Batch
where
```

```
    Conn: Into<String>,
    M: Into<String>,
    C: Currency,
{
    let mut batch = Batch::default();

    batch.schedule_execute_no_reply(NeutronMsg::submit_tx(
        connection.into(),
        ICA_ACCOUNT_ID.into(),
        trx.into_msgs(),
        memo.into(),
        timeout.secs(),
        IbcFee {
            recv_fee: vec![],
            ack_fee: vec![coin_legacy::to_cosmwasm_impl(ack_tip)],
            timeout_fee: vec![coin_legacy::to_cosmwasm_impl(
↳ timeout_tip)],
        },
    ));
    batch
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

It is recommended to allow tipping for the reception of packets (recv_fee) in the IBC protocol by adding a recv_tip parameter to the submit_-transaction function and passing it to the IbcFee struct.

Remediation Plan:

ACKNOWLEDGED: The Nolus team acknowledged this finding, stating that:

> Neutron does not support recv_fee. We follow the Neutron's solution for tips/fees for the relayers until they eventually migrate to the ICS standard one.

4.5 (HAL-05) REDUNDANT FIELD IN THE LOAN STRUCT - INFORMATIONAL (0.0)

Description:

The `loan` and its `current_period` field both keep track of an `annual_margin_interest`. This is a duplicated value that can be removed, as duplicating data can lead to inconsistencies when updating one of the values but not the other.

Code Location:

- contracts/lease/src/loan/mod.rs#L78-L96

Listing 4

```
pub(super) fn new(
    start: Timestamp,
    lpp_loan: LppLoan,
    annual_margin_interest: Percent,
    interest_payment_spec: InterestPaymentSpec,
) -> Self {
    let current_period = Self::due_period(
        annual_margin_interest,
        start,
        interest_payment_spec.due_period(),
    );
    Self {
        annual_margin_interest,
        lpn: PhantomData,
        lpp_loan,
        interest_payment_spec,
        current_period,
    }
}
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

It is recommended to remove the `annual_margin_interest` field from the `loan` struct and use the `annual_margin_interest` field from the `current_period` struct instead.

Remediation Plan:

SOLVED: The Nolus team has solved this issue in the commit ID `94fd-cdb01eceb25e9604b933fe48d835d90ca1c6`.

4.6 (HAL-06) DUPLICATE STORAGE READ - INFORMATIONAL (0.0)

Description:

The is_registered function in the Feeders struct reads from the storage twice. The first read is to check if the feeders field is None, and the second read is to get the feeders field. This can be refactored to read from the storage only once, since the may_load function returns an Option that can carry the feeders field if it exists. This applies to the remove function as well, where the update function reads from storage again.

Code Location:

- packages/marketprice/src/feeders.rs#L43-L51

Listing 5

```
pub fn is_registered(&self, storage: &dyn Storage, address: &Addr)
  -> StdResult<bool> {
    if self.0.may_load(storage)?.is_none() {
        return Ok(false);
    }

    let addrs = self.0.load(storage)?;

    Ok(addrs.contains(address))
}
```

- packages/marketprice/src/feeders.rs#L67-L78

Listing 6

```
pub fn remove(&self, deps: DepsMut<'_>, addr: Addr) -> Result<(),
PriceFeedersError> {
    let remove_address = |mut addrs: HashSet<Addr>| -> StdResult<
HashSet<Addr>> {
```

```
3           addrs.remove(&addr);
4           Ok(addrs)
5       };
6
7       if self.0.may_load(deps.storage)?.is_some() {
8           self.0.update(deps.storage, remove_address)?;
9       }
10
11      Ok(())
12  }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

It is recommended to refactor the is_registered function to read from storage only once.

Remediation Plan:

SOLVED: The Nolus team has solved this issue in the commit ID efcb75d93385e46b44fe420ee264315e8bbb347c for the is_registered function and the commit ID 55cfe785d29eb8b29d5082c24254f10ddbb68b8a for the remove function.

4.7 (HAL-07) OUTDATED OBSERVATIONS CAN BE STORED - INFORMATIONAL (0.0)

Description:

The add_observation function in the Feed struct adds an observation to the observations field. However, there is no check to ensure that the at field of the observation is greater than or equal to the valid_since field. This can lead to outdated observations being stored in the observations field.

Note that the outdated observations are filtered out when the calc_price function is called.

Code Location:

- packages/marketprice/src/feed/mod.rs#L33-L45

Listing 7

```
pub fn add_observation(
    mut self,
    from: Addr,
    at: Timestamp,
    price: Price<C, QuoteC>,
    valid_since: Timestamp,
) -> Self {
    self.observations
        .retain(observation::valid_since(valid_since));

    self.observations.push(Observation::new(from, at, price));
    self
}
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

It is recommended to add a check to ensure that the `at` field of the observation is greater than or equal to the `valid_since` field.

Remediation Plan:

SOLVED: The Nolus team has solved this issue in the commit ID `456837837af0232651dfe6e0955ddbd3d7ee1102`.

4.8 (HAL-08) RELAYER TIPS ARE HARDCODED - INFORMATIONAL (0.0)

Description:

Interchain Standard 29 (ICS 29), also known as “relayer fees”, is a feature aimed at incentivizing Inter-Blockchain Communication (IBC) infrastructure within the Cosmos ecosystem. ICS 29 is designed to make the job of relayers more sustainable by providing an incentivization layer that rewards relayer operators for successfully relaying packets. The standard defines three types of relayers: Forward relayer, who submits the recv_packet message for a given packet; Reverse relayer, who submits the ack_packet message for a given packet; and Timeout relayer, who submits the timeout_packet message for a given packet

It was found that the relayer tips are hardcoded in the packages/dex/src/trx.rs file. This means that the relayer tips cannot be changed without a code change. Currently, the relayer tips are set to 1unls which is not ideal since the relayer tips should be set to a value that is appropriate for the network.

Code Location:

- packages/dex/src/trx.rs#L21-L27

Listing 8

```
//TODO take them as input from the client
const ICA_TRANSFER_ACK_TIP: Coin<Nls> = Coin::new(1);
const ICA_TRANSFER_TIMEOUT_TIP: Coin<Nls> = ICA_TRANSFER_ACK_TIP;

//TODO take them as input from the client
const ICA_SWAP_ACK_TIP: Coin<Nls> = Coin::new(1);
const ICA_SWAP_TIMEOUT_TIP: Coin<Nls> = ICA_SWAP_ACK_TIP;
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

It is recommended to take the relayer tips as input from the client. Note that the TODO comments in the code snippet above indicate that this is already planned.

Remediation Plan:

ACKNOWLEDGED: The Nolus team acknowledged this finding.

MANUAL TESTING

In the manual testing phase, the following scenarios were simulated. The scenarios listed below were selected based on the severity of the vulnerabilities Halborn was testing the program for.

5.1 VERIFYING ACCESS CONTROL ON ENTRYPOINTS

Access control often represents a vulnerable point in smart contract assessments. To ensure that privileged operations are only accessible by designated users, the following tests were performed:

1. Contract access control: Verify that the privileged entrypoints requiring access control could only be invoked by the specified contract. For instance, the OpenLoan entrypoint in the LPP contract was accessible only by the Leaser.

```
ziion@ziion:~/Cosmos/nolus/nolus-core$ nolusd tx wasm execute $LPP '{"open_loan":{"amount":{"amount
": "10000", "ticker":"USDC"}}}' --amount 10000ibc/E2341B75AC852F89DF665F950D1C6F3EA57D69C4FEBF5ABC4
2E50F2468B7CD0F --from alice --keyring-backend test --node "tcp://0.0.0.0:26612" --fees 1000unls --
gas auto --gas-adjustment 1.3 -y -o json
Error: rpc error: code = Unknown desc = rpc error: code = Unknown desc = failed to execute message;
 message index: 0: [Lpp] [Platform] [Std] Generic error: Querier system error: No such contract: no
lus1qxy69294hn3lxw9tjd5fensdhaewwrkpyqzjm3: execute wasm contract failed [CosmWasm/wasmd@v0.31.0/x/
wasm/keeper/keeper.go:376] With gas wanted: '0' and gas used: '163902' : unknown request
```

Figure 1: Try OpenLoan on LPP

```
ziion@ziion:~/Cosmos/nolus/nolus-core$ nolusd tx wasm execute $TREASURY '{"send_rewards":{"amount": {"amount":"100"
, "ticker":"USDC"}}}' --from bob --keyring-backend test --node "tcp://0.0.0.0:26612" --fees 1750unls --gas auto --g
as-adjustment 1.3 -y -o json
Error: rpc error: code = Unknown desc = rpc error: code = Unknown desc = failed to execute message; message index:
0: [Treasury] [Access Control] Checked address doesn't match the one associated with access control variable!: exec
ute wasm contract failed [CosmWasm/wasmd@v0.31.0/x/wasm/keeper/keeper.go:376] With gas wanted: '0' and gas used: '1
36721' : unknown request
```

Figure 2: Try SendRewards on Treasury

2. User access control: Verify that the entrypoints requiring access control could only be invoked by the specified user. For example, only the user who initially opened a loan could close it or only users who have made a deposit can claim rewards.

```
ziion@ziion:~/Cosmos/nolus/nolus-core$ nolusd tx wasm execute $LPP '{"claim_rewards":{"other_recipient":"'$BOB'"}}'
 --from bob --keyring-backend test --node "tcp://0.0.0.0:26612" --fees 1000unls --gas auto --gas-adjustment 1.3 -y
-o json
Error: rpc error: code = Unknown desc = rpc error: code = Unknown desc = failed to execute message; message index:
0: [Lpp] The deposit does not exist: execute wasm contract failed [CosmWasm/wasmd@v0.31.0/x/wasm/keeper/keeper.go:3
76] With gas wanted: '0' and gas used: '136709' : unknown request
```

Figure 3: Claim rewards from other user

3. Configuration updates: Validate that all entrypoints that update or modify configuration parameters could only be invoked by the contract owner or a privileged user.

```
ziion@ziion:~/Cosmos/nolus/nolus-core$ nolusd tx wasm execute $LPP '{"new_lease_code":{"lease_code_id":"10"}}' --fr
om alice --keyring-backend test --node "tcp://0.0.0.0:26612" --fees 1000unls --gas auto --gas-adjustment 1.3 -y -o
json
Error: rpc error: code = Unknown desc = rpc error: code = Unknown desc = failed to execute message; message index:
0: [Lpp] [Access Control] Checked address doesn't match the one associated with access control variable!: execute w
asm contract failed [CosmWasm/wasmd@v0.31.0/x/wasm/keeper/keeper.go:376] With gas wanted: '0' and gas used: '136426
' : unknown request
```

Figure 4: Try to update Lease code

5.2 VERIFYING ERROR HANDLING DURING CONTRACT INSTANTIATION

To mitigate potential unintentional errors leading to misconfigurations during contract instantiation, the following tests were performed:

1. Address validation: Verify that addresses are validated before being stored in the configuration.
2. Rates validation: Ensure that the input parameters defined as rates were within the limits of 0%-100%. In the case of the Leaser, it has been also checked the order of the different rates in the liability (e.g: first warning < max).

```
{"level":"debug","module":"x/wasm","capabilities":"iterator,neutron,staking,stargate","cod
e_id":3,"time":"2023-07-10T19:42:30+02:00","message":"storing new contract"}
panic: genesis: Error parsing into type lpp::msg::InstantiateMsg: Rates should not be grea
ter than a hundred percent!: instantiate wasm contract failed

goroutine 1 [running]:
```

Figure 5: Borrow rate greater than 100%

```
{"level":"debug","module":"x/wasm","capabilities":"iterator,neutron,staking,stargate","cod
e_id":7,"time":"2023-07-10T19:35:54+02:00","message":"storing new contract"}
panic: genesis: Error parsing into type leaser::msg::InstantiateMsg: [Finance] Programming
 error or invalid serialized object of 'finance::liability::Liability' type, cause 'Initia
l % should be <= healthy %': instantiate wasm contract failed

goroutine 1 [running]:
```

Figure 6: Liability check

3. No empty objects: Ensure that those objects that are not Optional have valid values (e.g: tvl_to_apr could not contain repeated or empty values).

```
{"level":"debug","module":"x/wasm","capabilities":"iterator,neutron,staking,stargate","cod
e_id":8,"time":"2023-07-10T19:51:37+02:00","message":"storing new contract"}
panic: genesis: Error parsing into type rewards_dispatcher::msg::InstantiateMsg: Generic e
rror: Argument vector contains duplicates!: instantiate wasm contract failed

goroutine 1 [running]:
```

Figure 7: No duplicated values

```
{"level":"debug","module":"x/wasm","capabilities":"iterator,neutron,staking,stargate","cod
e_id":8,"time":"2023-07-10T19:40:25+02:00","message":"storing new contract"}
panic: genesis: Error parsing into type rewards_dispatcher::msg::InstantiateMsg: Generic e
rror: Argument vector contains no bars!: instantiate wasm contract failed

goroutine 1 [running]:
```

Figure 8: No empty values

5.3 VERIFYING INPUT VALIDATION ON THE ENTRY POINTS

Entrypoints lacking access control for privileged accounts/contracts can be executed by any user, allowing any input value. To test the contracts' behavior under these circumstances, the following tests were conducted:

1. User Input Validation: Verify that all user inputs in the entrypoints were checked before being stored or used in operations. For instance, trying to open a loan with a currency not registered in the PaymentGroup.

```
ziion@ziion:~/Cosmos/nolus/nolus-core$ nolusd tx wasm execute $LEASER '{"open_lease":{"currency": "NLS" ,"max_ltd":
500}}' --amount 2000000ibc/E2341B75AC852F89DF665F950D1C6F3EA57D69C4FEBF5ABC42E50F2468B7CD0F --from bob --keyring-ba
ckend test --node "tcp://0.0.0.0:26612" --fees 2800unls --gas auto --gas-adjustment 1.3 -y -o json
Error: rpc error: code = Unknown desc = rpc error: code = Unknown desc = failed to execute message; message index:
0: dispatch: submessages: [Lease] [Finance] Found currency 'NLS' which is not defined in the lease currency group:
instantiate wasm contract failed [CosmWasm/wasmd@v0.31.0/x/wasm/keeper/keeper.go:307] With gas wanted: '0' and gas
used: '267348' : unknown request
```

Figure 9: Try OpenLease with wrong currency

2. Configuration updates: Verify that the update configuration entrypoints perform the corresponding input validation, as well as during the instantiation.

```
ziion@ziion:~/Cosmos/nolus/nolus-core$ nolusd tx gov submit-proposal sudo-contract $LEASER '{"config":{"lease_inter
est_rate_margin":300, "liability":{"initial":6500,"healthy":700,"first_liq_warn":720,"second_liq_warn":750,"third_l
iq_warn":780,"max":800,"recalc_time":3600000000000}, "lease_interest_payment":{"due_period":3600000000000,"grace_pe
riod":1800000000000}}}' --title "Update Config" --description "Update Config" --deposit 10000000unls --fees 900unls
 --gas auto --gas-adjustment 1.1 --from reserve
Error: rpc error: code = Unknown desc = rpc error: code = Unknown desc = failed to execute message; message index:
0: Error parsing into type leaser::msg::SudoMsg: [Finance] Programming error or invalid serialized object of 'finan
ce::liability::Liability' type, cause 'Initial % should be <= healthy %': execute wasm contract failed: invalid pro
posal content [cosmos/cosmos-sdk@v0.45.15/x/gov/keeper/proposal.go:24] With gas wanted: '0' and gas used: '138275'
: unknown request
```

Figure 10: Update input validation

5.4 VERIFYING REPEATED OPERATIONS

Executing entrypoints in an unexpected order or more than once can lead to unpredictable behavior. To address this, the following tests were performed:

1. Multiple Rewards distribution before claim (out of the Alarm system): Verify that the number of rewards assigned to each lender remained accurate even when distributed at different moments.
2. Dex setup repetition: Confirm that the DEX, once set up, could not be modified.

```
ziion@ziion:~/Cosmos/nolus/nolus-core$ nolusd tx gov submit-proposal sudo-contract $LEASER '{"setup_dex": {"connect
ion_id": "connection-0", "transfer_channel": {"local_endpoint": "channel-0", "remote_endpoint": "channel-409"}}}' -
-title "Set up the DEX parameter" --description "The proposal aims to set the DEX parameters in the Leaser contract
" --deposit 10000000unls --fees 900unls --gas auto --gas-adjustment 1.1 --from reserve
Error: rpc error: code = Unknown desc = rpc error: code = Unknown desc = failed to execute message; message index:
0: [Leaser] DEX connectivity already setup: execute wasm contract failed: invalid proposal content [cosmos/cosmos-s
dk@v0.45.15/x/gov/keeper/proposal.go:24] With gas wanted: '0' and gas used: '140717' : unknown request
```

Figure 11: Dex setup repetition

3. Loan repayment and closure: Confirm that a loan could not be closed before being repaid and that it could not be repaid or closed twice

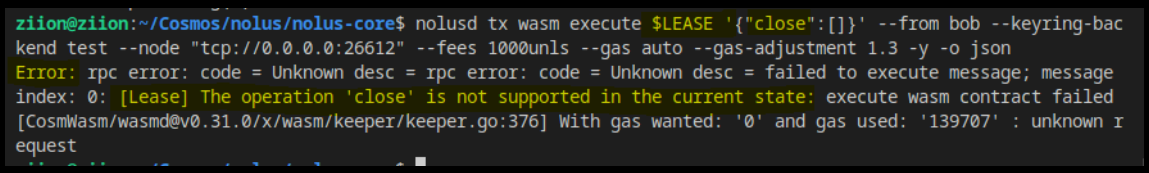

Figure 12: Try to close an unpaid loan

AUTOMATED TESTING

6.1 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. 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.

Listing 9: Dependency tree

```
No issues were detected.
```

THANK YOU FOR CHOOSING

// HALBORN