// HALBORN

Kryptonite - Protocol

CosmWasm Smart Contract Security Assessment

Prepared by: **Halborn**

Date of Engagement: **August 28th, 2023 - November 1st, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE
0.1	Document Creation	08/28/2023
0.2	Document Updates	10/31/2023
0.3	Document Updates	11/01/2023
0.4	Document Updates	11/02/2023
0.5	Document Updates	11/03/2023
0.6	Draft Version	11/06/2023
0.7	Draft Review	11/07/2023
1.0	Remediation Plan	11/22/2023
1.1	Remediation Plan Review	11/24/2023

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
Elena Maranon	Halborn	elena.maranon@halborn.com
Alexis Fabre	Halborn	alexis.fabre@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Kryptonite engaged Halborn to conduct a security assessment on their smart contracts beginning on August 28th, 2023 and ending on November 1st, 2023. The security assessment was scoped to the smart contracts provided to the Halborn team.

1.2 ASSESSMENT SUMMARY

The team at Halborn assigned two full-time security engineers to verify the security of the smart contracts. 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 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, which were mostly addressed by the Kryptonite team. The main ones were the following:

- Add an adequate access control in the central_control, custody, custody_base, custody_bsei, overseer and stable_pool contracts.
- Decrease users balance for each liquidated collateral.
- Update the calculation of the maximum loan-to-value when minting coins.
- Query the balance of the stable_pool contract when liquidating collaterals.
- Add the "!" symbol when validating the bid and liquidation fees.
- Update the calculation of the maximum loan-to-value when querying about available collaterals.
- Include an execution message to bond rewards.

- Include an execution message to swap all native tokens to reward denomination.
- Validate the balance change when processing the withdraw rate.
- Handle a list with pending redelegations and implement a public function to trigger the redelegation process over this list.
- Verify if there is any bid already submitted before updating the maximum amount of slots.

1.3 SCOPE

CosmWasm Smart Contracts in scope:

1. Kryptonite bAsset Contracts:
   (a) Repository: krp-basset-convert
   (b) Commit ID: 56c7ccf
   (c) Contracts in scope:
      - krp_basset_converter
      - krp_basset_token
2. Kryptonite CDP Contracts:
   (a) Repository: krp-cdp-contracts
   (b) Commit ID: 8023786
   (c) Contracts in scope:
      - central_control
      - custody
      - liquidation_queue
      - reward_book
      - stable_pool
3. Kryptonite Market Contracts:
   (a) Repository: krp-market-contracts
   (b) Commit ID: bbaafdc
   (c) Contracts in scope:
      - custody_base
      - custody_bsei
      - distribution_model
      - interest_model
      - liquidation_queue
      - market
      - overseer
4. Kryptonite Oracle:
   (a) Repository: krp-oracle
   (b) Commit ID: 96de964

(c) Contract in scope:

- oracle_pyth

5. Kryptonite Staking Contracts:

(a) Repository: krp-staking-contracts
(b) Commit ID: f58a4d4
(c) Contracts in scope:

- basset_sei_hub
- basset_sei_reward
- basset_sei_rewards_dispatcher
- basset_sei_token_bsei
- basset_sei_token_stsei
- basset_sei_validators_registry

6. Kryptonite Token Contracts:

(a) Repository: krp-token-contracts
(b) Commit ID: 1b55b39
(c) Contracts in scope:

- boost
- dispatcher
- distribute
- fund
- keeper
- seilor
- staking
- treasure
- ve_seilor

7. Swap Extension:

(a) Repository: swap-extension
(b) Commit ID: 2aeb0a7
(c) Contract in scope:

- swap_sparrow

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 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)

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
9	7	6	18	13

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) LOANS CAN BE REPAID WITHOUT SPENDING COINS	Critical (10)	SOLVED - 10/18/2023
(HAL-02) ARBITRARY MINTING OF COINS WITHOUT DEPOSITING COLLATERALS	Critical (10)	SOLVED - 10/18/2023
(HAL-03) NO ACCESS CONTROL IN WITHDRAW FUNCTION	Critical (10)	SOLVED - 10/21/2023
(HAL-04) NO ACCESS CONTROL IN REPAYMENT ENTRY POINT	Critical (10)	SOLVED - 10/21/2023
(HAL-05) ARBITRARY MINTING OF COINS USING FAKE CW20 TOKENS	Critical (10)	SOLVED - 10/18/2023
(HAL-06) COLLATERAL BALANCE OF LIQUIDATED USERS DOES NOT DECREASE	Critical (10)	SOLVED - 10/19/2023
(HAL-07) LIQUIDATED LOANS WITHOUT AN ADEQUATE REPAYMENT	Critical (10)	SOLVED - 10/18/2023
(HAL-08) ARBITRARY REPAYMENT OF COINS FROM LIQUIDATIONS	Critical (10)	SOLVED - 10/18/2023
(HAL-09) MISCALCULATION OF MAX LOAN TO VALUE WHEN MINTING COINS	Critical (9.4)	SOLVED - 10/18/2023
(HAL-10) COLLATERAL LIQUIDATION IS NOT WORKING AS EXPECTED	High (7.5)	SOLVED - 10/18/2023
(HAL-11) BID FEE IS NOT SENT	High (7.5)	SOLVED - 10/19/2023
(HAL-12) LIQUIDATOR FEE IS NOT SENT	High (7.5)	SOLVED - 10/19/2023
(HAL-13) MISCALCULATION OF MAX LOAN TO VALUE WHEN QUERYING AVAILABLE COLLATERAL	High (7.5)	SOLVED - 10/18/2023
(HAL-14) REWARDS BONDING FUNCTIONALITY IS UNAVAILABLE	High (7.5)	SOLVED - 10/19/2023
(HAL-15) WITHOUT POSSIBILITY TO SWAP ALL NATIVE TOKENS TO REWARD DENOMINATION	High (7.5)	SOLVED - 10/19/2023
(HAL-16) UNCHECKED BALANCE CHANGE COULD LEAD TO UNFAIR WITHDRAWALS	High (7.1)	SOLVED - 10/19/2023

(HAL-17) INADEQUATE TRACKING OF PENDING REDELEGATIONS	Medium (6.2)	SOLVED - 10/19/2023
(HAL-18) MAX NUMBER OF BID SLOTS COULD BE MODIFIED AFTER BIDS WERE SUBMITTED	Medium (6.2)	RISK ACCEPTED
(HAL-19) SOME FUNCTIONS RECEIVE MULTIPLE NATIVE COINS INSTEAD OF ONE	Medium (5.0)	SOLVED - 10/24/2023
(HAL-20) REDELEGATION IS NOT RESTRICTED TO ACTIVE VALIDATORS	Medium (5.0)	SOLVED - 10/20/2023
(HAL-21) COIN DENOMINATION IS NOT CHECKED WHEN REMOVING VALIDATORS	Medium (5.0)	RISK ACCEPTED
(HAL-22) INADEQUATE TRACKING OF SWAPPED AMOUNTS	Medium (5.0)	SOLVED - 10/23/2023
(HAL-23) UNBOND WAIT LISTS CAN BE MIGRATED EVEN IF THE CONTRACT IS NOT PAUSED	Low (3.4)	SOLVED - 10/20/2023
(HAL-24) UNCHECKED MAX LOAN-TO-VALUE RATIO	Low (3.4)	SOLVED - 10/19/2023
(HAL-25) FUNCTIONALITY TO UPDATE FEES IS NOT APPLIED CORRECTLY	Low (3.4)	SOLVED - 10/20/2023
(HAL-26) ARBITRARY MESSAGES CAN BE EXECUTED ON BEHALF OF THE HUB	Low (3.4)	SOLVED - 10/20/2023
(HAL-27) UNRELIABLE SOURCE OF RANDOMNESS	Low (3.3)	SOLVED - 10/30/2023
(HAL-28) WITHDRAWAL COULD GET STUCK IF THERE ARE NO MORE UNBONDINGS	Low (3.1)	RISK ACCEPTED
(HAL-29) OWNERSHIP CAN BE TRANSFERRED WITHOUT CONFIRMATION	Low (2.5)	SOLVED - 10/26/2023
(HAL-30) COMMENTED TRANSACTION MESSAGE	Low (2.5)	SOLVED - 11/21/2023
(HAL-31) UNCHECKED REDEEM FEE	Low (2.1)	SOLVED - 10/18/2023
(HAL-32) UNCHECKED KEEPER RATE	Low (2.1)	SOLVED - 10/20/2023
(HAL-33) MAXIMUM AMOUNT OF TOKENS TO MINT IS NOT VALIDATED	Low (2.1)	SOLVED - 10/25/2023

(HAL-34) UNCHECKED PARAMETERS IN DISPATCHER CONTRACT	Low (2.1)	SOLVED - 10/25/2023
(HAL-35) UNCHECKED PARAMETERS IN TREASURE CONTRACT	Low (2.1)	SOLVED - 10/30/2023
(HAL-36) DURATION IS NOT VALIDATED IN STAKING CONTRACT	Low (2.1)	SOLVED - 10/30/2023
(HAL-37) CLAIMABLE TIME IS NOT VALIDATED IN FUND CONTRACT	Low (2.1)	SOLVED - 10/31/2023
(HAL-38) EXCHANGE RATE COULD INCREASE INDEFINITELY	Low (2.1)	SOLVED - 10/20/2023
(HAL-39) PAIR KEY CAN CONTAIN DUPLICATED ASSETS	Low (2.1)	SOLVED - 10/23/2023
(HAL-40) ASSETS COULD MISMATCH WITH THE ONES IN PAIR ADDRESS	Low (2.1)	SOLVED - 10/24/2023
(HAL-41) IMMUTABLE VARIABLES CAN BE CHANGED IN STAKING CONTRACT	Informational (1.7)	SOLVED - 10/30/2023
(HAL-42) MARKETING INFO IS NOT VALIDATED AT INSTANTIATION	Informational (1.7)	SOLVED - 10/25/2023
(HAL-43) LOCK AMOUNT IN GLOBAL STATE IS NOT VALIDATED WHEN CLAIMING	Informational (1.7)	SOLVED - 10/25/2023
(HAL-44) UNCHECKED VALIDATOR ADDRESSES	Informational (1.2)	SOLVED - 10/20/2023
(HAL-45) UNCHECKED SAFE RATIO	Informational (0.8)	SOLVED - 10/19/2023
(HAL-46) REDUNDANT LOGIC	Informational (0.0)	SOLVED - 10/21/2023
(HAL-47) REPEATED EXECUTION MESSAGES	Informational (0.0)	ACKNOWLEDGED
(HAL-48) STAKING TOKEN CALLS CHECK SLASHING TWICE	Informational (0.0)	SOLVED - 10/24/2023
(HAL-49) UNIMPLEMENTED MESSAGE	Informational (0.0)	SOLVED - 10/20/2023
(HAL-50) UNUSED MESSAGES	Informational (0.0)	SOLVED - 10/25/2023

(HAL-51) UNUSED VARIABLES	Informational (0.0)	SOLVED - 10/30/2023
(HAL-52) USELESS FUNCTIONS	Informational (0.0)	ACKNOWLEDGED
(HAL-53) HARDCODED DENOM	Informational (0.0)	SOLVED - 10/21/2023

FINDINGS & TECH DETAILS

4.1 (HAL-01) LOANS CAN BE REPAID WITHOUT SPENDING COINS - CRITICAL(10)

Description:

The RepayStableCoin entry point of the **krp-cdp-contracts/central_control** contract allows users to repay their loans.

The entry point mentioned is designed to be called by the **krp-cdp-contracts/stable_pool** contract. However, there is no access control in the repay_stable_coin function to verify this condition.

As a consequence, anyone can call this function to repay their loans without spending coins.

Code Location:

There is no access control in the repay_stable_coin function:

Listing 1: krp-cdp-contracts/contracts/central_control/src/contract.rs (Line 612)

```
610 pub fn repay_stable_coin(
611  deps: DepsMut,
612  _info: MessageInfo,
613  sender: String,
614  amount: Uint128,
615 ) -> Result<Response, ContractError> {
616  let minter_raw = deps.api.addr_canonicalize(&sender.as_str())?;
617  let mut loan_info = read_minter_loan_info(deps.storage, &
 ↳ minter_raw)?;
618  loan_info.loans = loan_info.loans - Uint256::from(amount);
619  store_minter_loan_info(deps.storage, &minter_raw, &loan_info)?;
620
621  Ok(Response::new().add_attributes(vec![
622   attr("contract_name", "central_control"),
623   attr("action", "repay_stable_coin"),
```

```
624     attr("sender", sender.to_string()),
625     attr("amount", amount.to_string()),
626   ]))
627 }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:C/Y:N/R:N/S:U (10)

Recommendation:

Add an access control in the repay_stable_coin function to ensure that only the **stable_pool** contract can access the entry point mentioned.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commit 64d3a2d.

4.2 (HAL-02) ARBITRARY MINTING OF COINS WITHOUT DEPOSITING COLLATERALS - CRITICAL(10)

Description:

The MintStableCoin entry point of the **krp-cdp-contracts/central_control** contract mints stable coins to users according to their deposited collaterals.

The entry point mentioned is designed to be called by the **krp-cdp-contracts/custody** contract. However, there is no access control in the mint_stable_coin function to verify this condition.

As a consequence, anyone can call this function to mint stable coins to himself/herself without depositing collaterals.

Code Location:

There is no access control in the mint_stable_coin function:

Listing 2: krp-cdp-contracts/contracts/central_control/src/contract.rs (Line 512)

```
510 pub fn mint_stable_coin(
511  deps: DepsMut,
512  _info: MessageInfo,
513  minter: String,
514  stable_amount: Uint128,
515  collateral_amount: Option<Uint128>,
516  collateral_contract: Option<String>,
517  is_redemption_provider: Option<bool>,
518 ) -> Result<Response, ContractError> {
519  let config = read_config(deps.as_ref().storage)?;
520  let api = deps.api;
521
522  let minter_raw = api.addr_canonicalize(minter.as_str())?;
```

```
523  let mut cur_collaterals: Tokens = read_collaterals(deps.storage,
 ↳ &minter_raw);
524
525  let mut messages: Vec<CosmosMsg> = vec![];
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:C/Y:N/R:N/S:U (10)

Recommendation:

Add an access control in the mint_stable_coin function to ensure that only the **custody** contract can access the entry point mentioned.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commit 4c5310b.

4.3 (HAL-03) NO ACCESS CONTROL IN WITHDRAW FUNCTION - CRITICAL(10)

Description:

The WithdrawCollateral entry point of the **krp-market-contracts/custody_base** and **krp-market-contracts/custody_bsei** contracts allows a borrower to withdraw any amount of previously deposited collateral.

This entry point is designed to be called by the **krp-market-contracts/overseer** contract; however, there is no access control in the withdraw_collateral function to verify that.

As a consequence, any attacker could withdraw any amount of collateral from any other user, since the borrower and amount values are taken from the entry point input.

Code Location:

Code fragment of the withdraw_collateral function of the **krp-market-contracts/custody_bsei** contract:

Listing 3: krp-market-contracts/contracts/custody_bsei/src/collateral.rs

```
60 pub fn withdraw_collateral(
61     deps: DepsMut,
62     borrower: String,
63     amount: Option<Uint256>,
64 ) -> Result<Response, ContractError> {
65     let config: Config = read_config(deps.storage)?;
66
67     let borrower_raw = deps.api.addr_canonicalize(borrower.as_str
↳ ())?;
68     let mut borrower_info: BorrowerInfo = read_borrower_info(deps.
↳ storage, &borrower_raw);
69
70     // Check spendable balance
71     let amount = amount.unwrap_or(borrower_info.spendable);
```

```
72      if borrower_info.spendable < amount {
73          return Err(ContractError::WithdrawAmountExceedsSpendable(
74              borrower_info.spendable.into(),
75          ));
76      }
77
78      // decrease borrower collateral
79      borrower_info.balance = borrower_info.balance - amount;
80      borrower_info.spendable = borrower_info.spendable - amount;
81
82      if borrower_info.balance == Uint256::zero() {
83          remove_borrower_info(deps.storage, &borrower_raw);
84      } else {
85          store_borrower_info(deps.storage, &borrower_raw, &
↳ borrower_info)?;
86      }
87
88      Ok(Response::new()
89          .add_message(CosmosMsg::Wasm(WasmMsg::Execute {
90              contract_addr: deps
91                  .api
92                  .addr_humanize(&config.collateral_token)?
93                  .to_string(),
94              funds: vec![],
95              msg: to_binary(&Cw20ExecuteMsg::Transfer {
96                  recipient: borrower.to_string(),
97                  amount: amount.into(),
98              })?,
99          }))
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:C/Y:N/R:N/S:U (10)

Recommendation:

Add an access control in the withdraw_collateral function of the **Custody Base** and **Custody Bsei** contracts to ensure that **only** the **Overseer** contract can access this entry point.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commit 717dbe0.

4.4 (HAL-04) NO ACCESS CONTROL IN REPAYMENT ENTRY POINT - CRITICAL(10)

Description:

The RepayStableFromYieldReserve entry point of the **krp-market-contracts/overseer** contract allows a borrower to repay his debt using the stable currency reserve accumulated in the contract.

This entry point has no access control, consequently, any user could repay their own debt without spending any coin, using the balance of the **Overseer** contract if it is sufficient.

Code Location:

There is no access control in the repay_stable_from_yield_reserve function:

Listing 4: krp-market-contracts/contracts/overseer/src/collateral.rs (Line 242)

```
239 pub fn repay_stable_from_yield_reserve(
240     deps: DepsMut,
241     env: Env,
242     _info: MessageInfo,
243     borrower: Addr,
244 ) -> Result<Response, ContractError> {
245     let config: Config = read_config(deps.storage)?;
246     let market = deps.api.addr_humanize(&config.market_contract)?;
247     let borrow_amount_res: BorrowerInfoResponse =
 ↳ query_borrower_info(
248         deps.as_ref(),
249         market.clone(),
250         borrower.clone(),
251         env.block.height,
252     )?;
253     let borrow_amount = borrow_amount_res.loan_amount;
254
255     let prev_balance: Uint256 = query_balance(
256         deps.as_ref(),
```

```
257         market.clone(),
258         config.stable_denom.to_owned(),
259     )?;
260
261     let repay_messages = vec![
262         CosmosMsg::Bank(BankMsg::Send {
263             to_address: market.to_string(),
264             amount: vec![Coin {
265                 denom: config.stable_denom,
266                 amount: borrow_amount.into(),
267             }],
268         }),
269         CosmosMsg::Wasm(WasmMsg::Execute {
270             contract_addr: market.to_string(),
271             funds: vec![],
272             msg: to_binary(&MarketExecuteMsg::
 ↳ RepayStableFromLiquidation {
273                 borrower: borrower.to_string(),
274                 prev_balance,
275             })?,
276         }),
277     ];
278
279     Ok(Response::new().add_messages(repay_messages))
280 }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:C/Y:N/R:N/S:U (10)

Recommendation:

Add an access control in the repay_stable_from_yield_reserve function of the **Overseer** contract to prevent a random user from paying his debt from the contract reserve.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commit e43051d.

4.5 (HAL-05) ARBITRARY MINTING OF COINS USING FAKE CW20 TOKENS - CRITICAL(10)

Description:

The MintStableCoin entry point of the **krp-cdp-contracts/custody** contract mints stable coins to users according to their deposited collaterals.

The entry point mentioned is designed to be called by the **target collateral** contract. However, there is no access control in the mint_stable_coin function to verify this condition.

As a consequence, anyone could call this function to mint coins for himself/herself using a fake CW20 token.

Code Location:

There is no access control in the mint_stable_coin function:

Listing 5: krp-cdp-contracts/contracts/custody/src/contract.rs (Line 276)

```
274 pub fn mint_stable_coin(
275  deps: DepsMut,
276  _info: MessageInfo,
277  sender: String,
278  amount: Uint128,
279  stable_amount: Uint128,
280  is_redemption_provider: Option<bool>,
281 ) -> Result<Response, ContractError> {
282  let config = read_config(deps.as_ref().storage)?;
283  let api = deps.api;
284  let control_contract = api.addr_humanize(&config.control_contract
 ↳ )?.to_string();
285
286  let mut state = read_state(deps.as_ref().storage)?;
287  state.total_amount = state.total_amount + Uint256::from(amount);
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:C/Y:N/R:N/S:U (10)

Recommendation:

Add an access control in the mint_stable_coin function to ensure that only the **target collateral** contract can access the entry point mentioned.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commit 1e6a609.

4.6 (HAL-06) COLLATERAL BALANCE OF LIQUIDATED USERS DOES NOT DECREASE - CRITICAL(10)

Description:

In the liquidate_collateral function from the **krp-cdp-contracts/central_control** contract, when each collateral in the liquidation_amount vector is liquidated, the DecreaseBalance message is not called.

As a consequence, the collateral balance of the liquidated user won't decrease and will continue accruing rewards.

Code Location:

The DecreaseBalance message is not called in the liquidate_collateral function:

Listing 6: krp-cdp-contracts/contracts/central_control/src/contract.rs

```
280 for collateral in liquidation_amount {
281  if collateral.1 > Uint256::zero() {
282   let whitelist_elem = read_whitelist_elem(deps.storage, &
 ↳ collateral.0)?;
283   liquidation_messages.push(CosmosMsg::Wasm(WasmMsg::Execute {
284    contract_addr: deps
285     .api
286     .addr_humanize(&whitelist_elem.custody_contract)?
287     .to_string(),
288    funds: vec![],
289    msg: to_binary(&CustodyExecuteMsg::LiquidateCollateral {
290     liquidator: info.sender.to_string(),
291     amount: collateral.1.into(),
292    })?,
293   }))
294  }
295 }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:C/R:N/S:U (10)

Recommendation:

Include a call to the DecreaseBalance message in the liquidate_collateral function for each collateral that is liquidated.

Remediation Plan:

SOLVED:The Kryptonite team solved this issue in commit 4a00da9.

4.7 (HAL-07) LIQUIDATED LOANS WITHOUT AN ADEQUATE REPAYMENT - CRITICAL(10)

Description:

In the liquidate_collateral function from the **krp-cdp-contracts/custody** contract, there is no validation that info.sender is the **krp-cdp-contracts/central_control** contract. As a consequence, the coins to be used as repayment will only be transferred to the **krp-cdp-contracts/stable_pool** contract, without further logic to be executed. Later, an attacker could force the repayment of those coins to an arbitrary account, as shown in the following example:

1. The attacker calls liquidate_collateral function in the **custody** contract.
2. An amount of stable coins are transferred from the **liquidation_queue** contract to the **stable_pool** contract.
3. The attacker forces the repayment of coins to himself, as described in the vulnerability (HAL-08) ARBITRARY REPAYMENT OF COINS FROM LIQUIDATIONS.

Code Location:

There is no access control in the liquidate_collateral function:

Listing 7: krp-cdp-contracts/contracts/custody/src/contract.rs (Line 385)

```
383 pub fn liquidate_collateral(
384  deps: DepsMut,
385  _info: MessageInfo,
386  liquidator: Addr,
387  amount: Uint128,
388 ) -> Result<Response, ContractError> {
389  let config = read_config(deps.storage)?;
390
```

```
391  let mut state = read_state(deps.storage)?;
392  state.total_amount = state.total_amount - Uint256::from(amount);
393  store_state(deps.storage, &state)?;
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:C/Y:N/R:N/S:U (10)

Recommendation:

Add an access control in the liquidate_collateral function to ensure that only the **central_control** contract can execute it.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commit 2997c00.

4.8 (HAL-08) ARBITRARY REPAYMENT OF COINS FROM LIQUIDATIONS - CRITICAL(10)

Description:

In the repay_stable_from_liquidation function from the **krp-cdp-contracts/stable_pool** contract, there is no validation that info.sender is the **krp-cdp-contracts/central_control** contract. As a consequence, an attacker could repay coins to himself after liquidating other users' loans, as shown in the following example:

1. The attacker gets an extremely low loan.
2. The attacker liquidates some users and forces the transfer of coins to the **stable_pool** contract, as described in the vulnerability (HAL-08) ARBITRARY REPAYMENT OF COINS FROM LIQUIDATIONS.
3. The attacker calls repay_stable_from_liquidation function from the **stable_pool** contract using pre_balance = 0 and minter = <ATTACKER_ADDRESS>.
4. Because the loan for the attacker is extremely low, almost all the coins will be transferred from the **stable_pool** contract to the attacker address.

Code Location:

There is no access control in the repay_stable_from_liquidation function:

Listing 8: krp-cdp-contracts/contracts/stable_pool/src/contract.rs

```
214 pub fn repay_stable_from_liquidation(
215  deps: DepsMut,
216  env: Env,
217  info: MessageInfo,
218  minter: Addr,
219  pre_balance: Uint256,
220 ) -> Result<Response<SeiMsg>, ContractError> {
221  let config = read_config(deps.storage)?;
```

```
222
223  let cur_balance: Uint256 = query_balance(
224   deps.as_ref(),
225   env.contract.address.clone(),
226   config.stable_denom.to_string(),
227  )?;
228
229  let mut info = info;
230  info.sender = minter;
231
232  info.funds = vec![Coin {
233   denom: config.stable_denom,
234   amount: (cur_balance - pre_balance).into(),
235  }];
236
237  repay_stable_coin(deps, info)
238 }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:C/Y:N/R:N/S:U (10)

Recommendation:

Add an access control in the repay_stable_from_liquidation function to ensure that only the **central_control** contract can execute it.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commit 41ae7eb.

4.9 (HAL-09) MISCALCULATION OF MAX LOAN TO VALUE WHEN MINTING COINS - CRITICAL(9.4)

Description:

In the mint_stable_coin function from **krp-cdp-contracts/central_control** contract, the max loan to value is miscalculated because the collaterals_values is increased in every step of the loop instead of storing the temporal value. As a consequence, users can borrow more coins than they should.

Currently, the value of collaterals_values is:

$$collaterals\_values+ = Uint256 :: from(collateral.1) * price.emv\_price;$$

However, it should be:

$$collaterals\_values = Uint256 :: from(collateral.1) * price.emv\_price$$

Code Location:

Listing 9: krp-cdp-contracts/contracts/central_control/src/contract.rs (Line 575)

```
567 for collateral in cur_collaterals {
568  let price = query_price(
569   deps.as_ref(),
570   deps.api.addr_humanize(&config.oracle_contract)?,
571   api.addr_humanize(&collateral.0)?.to_string(),
572   "".to_string(),
573   None,
574  )?;
575  collaterals_values += Uint256::from(collateral.1) * price.
 ↳ emv_price;
576
577  let collateral_info = read_whitelist_elem(deps.storage, &
 ↳ collateral.0)?;
```

```
578  max_loan_to_value += collaterals_values * collateral_info.max_ltv
 ↳ ;
579 }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:H/A:N/D:H/Y:N/R:N/S:U (9.4)

Recommendation:

Update the calculation of collaterals_values according to the expression suggested above.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commit 9e26c73.

4.10 (HAL-10) COLLATERAL LIQUIDATION IS NOT WORKING AS EXPECTED - HIGH (7.5)

Description:

The liquidate_collateral function in the **krp-cdp-contracts/central_control** contract is querying its own balance instead of the balance of **krp-cdp-contracts/stable_pool** contract. As a consequence, some unexpected situations can happen:

- Trying to repay with an erroneous amount of stable coins.
- Collateral liquidation fails, and the operation is reverted.

Code Location:

Listing 10: krp-cdp-contracts/contracts/central_control/src/contract.rs (Line 257)

```
255 let pre_balance: Uint256 = query_balance(
256  deps.as_ref(),
257  env.contract.address.clone(),
258  config.stable_denom.to_string(),
259 )?;
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:M/A:M/D:M/Y:N/R:N/S:U (7.5)

Recommendation:

Update the logic of the liquidate_collateral function to query the balance of the **stable_pool** contract.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commit 2b1e9bb.

4.11 (HAL-11) BID FEE IS NOT SENT - HIGH (7.5)

Description:

The execute_liquidation function in the **krp-cdp-contracts/liquidation_queue** and **krp-market-contracts/liquidation_queue** contracts is responsible for executing the liquidation of the debt using the previously deposited bids.

There is an error in the code at the time of sending the bid_fee: the zero validation is incorrect, resulting in a situation where the fee will only be sent if it is equal to zero. This validation should incorporate the ! symbol before the condition to make sense.

Code Location:

Code fragment of the execute_liquidation function in the **krp-market-contracts/liquidation_queue** contract:

Listing 11: krp-market-contracts/contracts/liquidation_queue/src/bid.rs (Line 400)

```
389 let mut messages: Vec<CosmosMsg> = vec![CosmosMsg::Bank(BankMsg::
 ↳ Send {
390     to_address: repay_address,
391     amount: vec![deduct_tax(
392         deps.as_ref(),
393         Coin {
394             denom: config.stable_denom.clone(),
395             amount: repay_amount.into(),
396         },
397     )?],
398 })];
399
400 if bid_fee.is_zero() {
401     messages.push(CosmosMsg::Bank(BankMsg::Send {
402         to_address: fee_address,
403         amount: vec![deduct_tax(
```

```
404             deps.as_ref(),
405             Coin {
406                 denom: config.stable_denom.clone(),
407                 amount: bid_fee.into(),
408             },
409         )?],
410     }));
411 }
412 if liquidator_fee.is_zero() {
413     messages.push(CosmosMsg::Bank(BankMsg::Send {
414         to_address: liquidator,
415         amount: vec![deduct_tax(
416             deps.as_ref(),
417             Coin {
418                 denom: config.stable_denom.clone(),
419                 amount: liquidator_fee.into(),
420             },
421         )?],
422     }));
423 }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:H/R:N/S:U (7.5)

Recommendation:

It is recommended to add the `!` symbol before each zero validation in the `bid_fee`.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commits `b3c5629` and `d00ba8d`.

4.12 (HAL-12) LIQUIDATOR FEE IS NOT SENT - HIGH (7.5)

Description:

The execute_liquidation function in the **krp-cdp-contracts/liquidation_queue** and **krp-market-contracts/liquidation_queue** contracts is responsible for executing the liquidation of the debt using the previously deposited bids.

There is an error in the code at the time of sending the liquidator_fee: the zero validation is incorrect, resulting in a situation where the fee will only be sent if it is equal to zero. This validation should incorporate the ! symbol before the condition to make sense.

Code Location:

Code fragment of the execute_liquidation function of the **krp-market-contracts/liquidation_queue** contract:

Listing 12: krp-market-contracts/contracts/liquidation_queue/src/bid.rs (Line 412)

```
389 let mut messages: Vec<CosmosMsg> = vec![CosmosMsg::Bank(BankMsg::
 ↳ Send {
390     to_address: repay_address,
391     amount: vec![deduct_tax(
392         deps.as_ref(),
393         Coin {
394             denom: config.stable_denom.clone(),
395             amount: repay_amount.into(),
396         },
397     )?],
398 })];
399
400 if bid_fee.is_zero() {
401     messages.push(CosmosMsg::Bank(BankMsg::Send {
402         to_address: fee_address,
403         amount: vec![deduct_tax(
```

```
404             deps.as_ref(),
405             Coin {
406                 denom: config.stable_denom.clone(),
407                 amount: bid_fee.into(),
408             },
409         )?],
410     }));
411 }
412 if liquidator_fee.is_zero() {
413     messages.push(CosmosMsg::Bank(BankMsg::Send {
414         to_address: liquidator,
415         amount: vec![deduct_tax(
416             deps.as_ref(),
417             Coin {
418                 denom: config.stable_denom.clone(),
419                 amount: liquidator_fee.into(),
420             },
421         )?],
422     }));
423 }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:H/R:N/S:U (7.5)

Recommendation:

It is recommended to add the `!` symbol before each zero validation in the `liquidator_fee`.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commits `b3c5629` and `d00ba8d`.

4.13 (HAL-13) MISCALCULATION OF MAX LOAN TO VALUE WHEN QUERYING AVAILABLE COLLATERAL - HIGH (7.5)

Description:

In the query_collateral_available function in the **krp-cdp-contracts/central_control** contract, the value of max_loans_value is miscalculated, and its value is overwritten in each step of the for loop. As a consequence, the value returned when querying about the available collateral for users will be inaccurate.

Currently, the value of max_loans_value is:

$$max\_loans\_value = collateral.1 * price\_resp.emv\_price * collateral\_info.max\_ltv$$

However, it should be:

$$max\_loans\_value += collateral.1 * price\_resp.emv\_price * collateral\_info.max\_ltv$$

Code Location:

Listing 13: krp-cdp-contracts/contracts/central_control/src/contract.rs (Line 343)

```
329 for collateral in collaterals {
330  let collateral_info = read_whitelist_elem(deps.storage, &
 ↳ collateral.0)?;
331  let price_resp = query_price(
332   deps,
333   deps.api.addr_humanize(&config.oracle_contract)?,
334   deps.api.addr_humanize(&collateral.0)?.to_string(),
335   "".to_string(),
336   None,
337  )?;
338  if collateral.0 == collateral_raw {
339   collateral_amount = collateral.1;
340   collateral_price = price_resp.emv_price;
```

```
341   collateral_max_ltv = collateral_info.max_ltv;
342  } else {
343   max_loans_value = collateral.1 * price_resp.emv_price *
 ↳ collateral_info.max_ltv;
344  }
345 }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:H/A:N/D:N/Y:N/R:N/S:U (7.5)

Recommendation:

Update the calculation of max_loans_value according to the expression suggested above.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commit 0950837.

4.14 (HAL-14) REWARDS BONDING FUNCTIONALITY IS UNAVAILABLE - HIGH (7.5)

Description:

Executing the ExecuteMsg::BondRewards message in **krp-staking-contracts/basset_sei_hub** contract is restricted for only the **krp-staking-contracts/basset_sei_rewards_dispatcher** contract. However, there is no function in that contract that allows to execute the bond, which makes the rewards bonding functionality unavailable.

Code Location:

The **krp-staking-contracts/basset_sei_rewards_dispatcher** contract does not call the ExecuteMsg::BondRewards message:

Listing 14: krp-staking-contracts/contracts/basset_sei_rewards_dispatcher/src/contract.rs (Lines 58,68,69,89,92,95)

```
55 #[cfg_attr(not(feature = "library"), entry_point)]
56 pub fn execute(deps: DepsMut, env: Env, info: MessageInfo, msg:
↳ ExecuteMsg) -> StdResult<Response> {
57  match msg {
58   ExecuteMsg::SwapToRewardDenom {
59    bsei_total_bonded: bsei_total_mint_amount,
60    stsei_total_bonded: stsei_total_mint_amount,
61   } => execute_swap(
62    deps,
63    env,
64    info,
65    bsei_total_mint_amount,
66    stsei_total_mint_amount,
67   ),
68   ExecuteMsg::DispatchRewards {} => execute_dispatch_rewards(deps,
↳ env, info),
69   ExecuteMsg::UpdateConfig {
70    owner,
71    hub_contract,
```

```
72      bsei_reward_contract,
73      stsei_reward_denom,
74      bsei_reward_denom,
75      krp_keeper_address,
76      krp_keeper_rate,
77    } => execute_update_config(
78      deps,
79      env,
80      info,
81      owner,
82      hub_contract,
83      bsei_reward_contract,
84      stsei_reward_denom,
85      bsei_reward_denom,
86      krp_keeper_address,
87      krp_keeper_rate,
88    ),
89    ExecuteMsg::UpdateSwapContract { swap_contract } => {
90      update_swap_contract(deps, info, swap_contract)
91    }
92    ExecuteMsg::UpdateSwapDenom { swap_denom, is_add } => {
93      update_swap_denom(deps, info, swap_denom, is_add)
94    }
95    ExecuteMsg::UpdateOracleContract { oracle_contract } => {
96      update_oracle_contract(deps, info, oracle_contract)
97    }
98  }
99 }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:H/D:N/Y:N/R:N/S:U (7.5)

Recommendation:

Include a call to the ExecuteMsg::BondRewards message when dispatching rewards.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commit a682bc4.

4.15 (HAL-15) WITHOUT POSSIBILITY TO SWAP ALL NATIVE TOKENS TO REWARD DENOMINATION - HIGH (7.5)

Description:

Executing the ExecuteMsg::SwapToRewardDenom message in the **krp-staking-contracts/basset_sei_reward** contract is restricted for only the **krp-staking-contracts/basset_sei_rewards_dispatcher** contract. However, there is no function in that contract that allows to execute the swap, which makes not possible to swap all native tokens to reward denomination.

Code Location:

The **krp-staking-contracts/basset_sei_rewards_dispatcher** contract does not call the ExecuteMsg::SwapToRewardDenom message:

Listing 15: krp-staking-contracts/contracts/basset_sei_rewards_dispatcher/src/contract.rs (Lines 58,68,69,89,92,95)

```
55 #[cfg_attr(not(feature = "library"), entry_point)]
56 pub fn execute(deps: DepsMut, env: Env, info: MessageInfo, msg:
↳ ExecuteMsg) -> StdResult<Response> {
57  match msg {
58   ExecuteMsg::SwapToRewardDenom {
59    bsei_total_bonded: bsei_total_mint_amount,
60    stsei_total_bonded: stsei_total_mint_amount,
61   } => execute_swap(
62    deps,
63    env,
64    info,
65    bsei_total_mint_amount,
66    stsei_total_mint_amount,
67   ),
68   ExecuteMsg::DispatchRewards {} => execute_dispatch_rewards(deps,
↳ env, info),
69   ExecuteMsg::UpdateConfig {
```

```
70      owner,
71      hub_contract,
72      bsei_reward_contract,
73      stsei_reward_denom,
74      bsei_reward_denom,
75      krp_keeper_address,
76      krp_keeper_rate,
77    } => execute_update_config(
78      deps,
79      env,
80      info,
81      owner,
82      hub_contract,
83      bsei_reward_contract,
84      stsei_reward_denom,
85      bsei_reward_denom,
86      krp_keeper_address,
87      krp_keeper_rate,
88    ),
89    ExecuteMsg::UpdateSwapContract { swap_contract } => {
90      update_swap_contract(deps, info, swap_contract)
91    }
92    ExecuteMsg::UpdateSwapDenom { swap_denom, is_add } => {
93      update_swap_denom(deps, info, swap_denom, is_add)
94    }
95    ExecuteMsg::UpdateOracleContract { oracle_contract } => {
96      update_oracle_contract(deps, info, oracle_contract)
97    }
98  }
99 }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:H/D:N/Y:N/R:N/S:U (7.5)

Recommendation:

Include a call to the `ExecuteMsg::SwapToRewardDenom` message when dispatching rewards.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commit a5c858c.

4.16 (HAL-16) UNCHECKED BALANCE CHANGE COULD LEAD TO UNFAIR WITHDRAWALS - HIGH (7.1)

Description:

The process_withdraw_rate function in the **krp-staking-contracts/basset_sei_hub** contract is not validating that the value of hub_balance is greater than prev_hub_balance. As a consequence, some edge scenarios could lead to users receiving less than expected when withdrawing. The following scenarios could arise in this condition (non-exhaustive list):

- Hub contract is migrated and the remaining balance is transferred to another address.
- Unauthorized transfer of funds out of the hub.
- Slashing of validators.

Code Location:

Listing 16: krp-staking-contracts/contracts/basset_sei_hub/src/unbond.rs (Line 292)

```
292 let balance_change = SignedInt::from_subtraction(hub_balance,
 ↳ state.prev_hub_balance);
293 let actual_unbonded_amount = balance_change.0;
294
295 let mut bsei_unbond_ratio = Decimal256::zero();
296 if stsei_total_unbonded_amount + bsei_total_unbonded_amount >
 ↳ Uint256::zero() {
297  let stsei_unbond_ratio = Decimal256::from_ratio(
298   stsei_total_unbonded_amount.0,
299   (stsei_total_unbonded_amount + bsei_total_unbonded_amount).0,
300  );
301  bsei_unbond_ratio = Decimal256::one() - stsei_unbond_ratio;
302 }
303
304 let bsei_actual_unbonded_amount = Uint256::from(
 ↳ actual_unbonded_amount) * bsei_unbond_ratio;
```

```
// Use signed integer in case of some rogue transfers.
let bsei_slashed_amount =
 SignedInt::from_subtraction(bsei_total_unbonded_amount,
 ↳ bsei_actual_unbonded_amount);
let stsei_slashed_amount = SignedInt::from_subtraction(
 stsei_total_unbonded_amount,
 Uint256::from(actual_unbonded_amount) -
 ↳ bsei_actual_unbonded_amount,
);
```

BVSS:

AO:A/AC:L/AX:M/C:N/I:H/A:M/D:H/Y:N/R:N/S:U (7.1)

Recommendation:

Validate that the value of `hub_balance` is greater than `prev_hub_balance` in the `process_withdraw_rate` function before further execution.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commit `9c0d95a`.

4.17 (HAL-17) INADEQUATE TRACKING OF PENDING REDELEGATIONS - MEDIUM (6.2)

Description:

When executing the remove_validator function in the **krp-staking-contracts/basset_sei_validators_registry** contract and the redelegation was not possible (e.g.: because of can_redelegate.amount), the validator is removed from the storage, but there is not an adequate tracking of pending redelegations to be done later.

Code Location:

Listing 17: krp-staking-contracts/contracts/basset_sei_validators_registry/src/contract.rs (Lines 160-162)

```
155 if let Some(delegation) = delegated_amount {
156  // Terra core returns zero if there is another active
 ↳ redelegation
157  // That means we cannot start a new redelegation, so we only
 ↳ remove a validator from
158  // the registry.
159  // We'll do a redelegation manually later by sending
 ↳ RedelegateProxy to the hub
160  if delegation.can_redelegate.amount < delegation.amount.amount {
161   return StdResult::Ok(Response::new());
162  }
163
164  let (_, delegations) =
165   calculate_delegations(delegation.amount.amount, validators.
 ↳ as_slice())?;
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:M/D:M/Y:N/R:N/S:U (6.2)

Recommendation:

It is recommended to handle a list with pending redelegations and also have a public function (i.e.: accessible to any user) to trigger the redelegation process over this list.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commit ab8ef52.

4.18 (HAL-18) MAX NUMBER OF BID SLOTS COULD BE MODIFIED AFTER BIDS WERE SUBMITTED - MEDIUM (6.2)

Description:

The update_collateral_info function in the **krp-cdp-contracts/liquidation_queue** and **krp-market-contracts/liquidation_queue** contracts allows the owner to modify some parameters of the CollateralInfo struct at any time. One of these parameters is max_slot, the maximum number of slots in the liquidation queue in each collateral for future user bids.

If a user has submitted some bids in any of the last slots in the queue and the max_slot parameter is replaced with a smaller one, the bids will not be used in future liquidations because the for loop of the execute_liquidation function will not reach them. This operation can only be executed by the owner, but the code does not check if there are already bids submitted in the last positions before replacing the parameter.

It is worth mentioning that the bids are not lost, they could be refunded calling the retract_bid function, but it requires a spent of gas from the user and, if the user is not notified about the update, the bids would be stuck for an undetermined time, missing the opportunity to invest these bids in other liquidation queues.

Code Location:

Code fragment of the update_collateral_info function in the **krp-market-contracts/liquidation_queue** contract:

Listing 18: krp-market-contracts/contracts/liquidation_queue/src/contract.rs (Lines 305-310)

```
301 if let Some(bid_threshold) = bid_threshold {
302     collateral_info.bid_threshold = bid_threshold;
```

```
303 }
304
305 if let Some(max_slot) = max_slot {
306     // assert max slot does not exceed cap and max premium rate
 ↳ does not exceed 1
307     assert_max_slot(max_slot)?;
308     assert_max_slot_premium(max_slot, collateral_info.
 ↳ premium_rate_per_slot)?;
309     collateral_info.max_slot = max_slot;
310 }
311
312 // save collateral info
313 store_collateral_info(deps.storage, &collateral_token_raw, &
 ↳ collateral_info)?;
314
315 Ok(Response::new().add_attribute("action", "update_collateral_info
 ↳ "))
316 }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:M/Y:M/R:N/S:U (6.2)

Recommendation:

It is recommended to add a validation process before modifying the max_slot parameter in order to check if there is any bid already submitted in the slots that would disappear after the update. If the bid exists and the parameter needs to be updated, the refund should be done by the owner of the contract.

Remediation Plan:

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

4.19 (HAL-19) SOME FUNCTIONS RECEIVE MULTIPLE NATIVE COINS INSTEAD OF ONE - MEDIUM (5.0)

Description:

Some functions in the **krp-basset-convert/krp_basset_converter** and **swap-extension/swap_sparrow** contracts can receive multiple native coins, which would allow a double payment if users mistakenly send native coins different from the target one. The affected functions are the following:

- **krp-basset-convert/krp_basset_converter:** execute_convert_to_basset function
- **swap-extension/swap_sparrow:** swap_denom function

Code Location:

Code fragment of the execute_convert_to_basset function in the **krp-basset-convert/krp_basset_converter** contract:

Listing 19: krp-basset-convert/contracts/krp_basset_converter/src/contract.rs (Lines 114-120)

```
112 let coin_denom = config.native_denom.unwrap();
113
114 let coin = info
115     .funds
116     .iter()
117     .find(|x| x.denom == coin_denom && x.amount > Uint128::zero())
118     .ok_or_else(|| {
119       StdError::generic_err(format!("No {} assets are provided to
 ↳ deposit", coin_denom));
120     });
121
122 let denom_decimals = 6u8;
```

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 that the functions mentioned above only accept the target native coins and reverse the operations otherwise.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commits a840e64 and d089b97.

4.20 (HAL-20) REDELEGATION IS NOT RESTRICTED TO ACTIVE VALIDATORS - MEDIUM (5.0)

Description:

The execute_redelegate_proxy function in the **krp-staking-contracts/basset_sei_hub** contract does not restrict that redelegation is done only to active validators, which could create unexpected situations. For example, if the owner mistakenly delegates to a non-active validator and wants to fix it by redelegating again to an active validator, he will need to wait a cooldown period (because of consecutive redelegations) to carry out this task.

Code Location:

Listing 20: krp-staking-contracts/contracts/basset_sei_hub/src/contract.rs (Lines 239-248)

```
234 if sender_contract_addr != validators_registry_contract &&
 ↳ sender_contract_addr != conf.creator
235 {
236  return Err(StdError::generic_err("unauthorized"));
237 }
238
239 let messages: Vec<CosmosMsg> = redelegations
240  .into_iter()
241  .map(|(dst_validator, amount)| {
242   cosmwasm_std::CosmosMsg::Staking(StakingMsg::Redelegate {
243                 src_validator: src_validator.clone(),
244                 dst_validator,
245                 amount,
246   })
247  })
248  .collect();
```

BVSS:

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

Recommendation:

It is recommended to restrict that the redelegations are done only to active validators.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commit 7de9f39.

4.21 (HAL-21) COIN DENOMINATION IS NOT CHECKED WHEN REMOVING VALIDATORS - MEDIUM (5.0)

Description:

When removing a validator using the remove_validator function in the **krp-staking-contracts/basset_sei_validators_registry** contract, there is no check about coin denomination in the delegated amount (i.e.: coin to be redelegated). As a consequence, some unexpected situations could arise in the protocol, e.g.: removing a validator with delegated native coins different from the target one.

Code Location:

There is no check about coin denomination in the remove_validator function:

Listing 21: krp-staking-contracts/contracts/basset_sei_validators_registry/src/contract.rs (Line 173)

```
167 for i in 0..delegations.len() {
168  if delegations[i].is_zero() {
169   continue;
170  }
171  redelegations.push((
172   validators[i].address.clone(),
173   Coin::new(delegations[i].u128(), delegation.amount.denom.as_str
 ↳ ()),
174  ));
175 }
```

BVSS:

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

Recommendation:

It is recommended to check coin denomination when removing validators.

Remediation Plan:

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

4.22 (HAL-22) INADEQUATE TRACKING OF SWAPPED AMOUNTS - MEDIUM (5.0)

Description:

The swap_denom function in the **swap-extension/swap_sparrow** contract updates the values of swap_info.total_amount_in and swap_info.total_amount_out variables whenever there is a swap operation. However, those values have a mix of both coins involved in the swap, so after some operations, it won't be possible to identify how much comes from each one of the coins.

Code Location:

Listing 22: swap-extension/contracts/swap_sparrow/src/handler.rs (Lines 169-170)

```
160 // swap
161 let asset = Asset {
162  amount: payment.amount,
163  info: asset_infos[0].clone(),
164 };
165 let pair_address = pair_config.pair_address.clone();
166 let offer_asset = asset.clone();
167 let simulation_response = query_simulation(&deps.querier,
 ↳ pair_address.clone().to_string(), offer_asset.clone())?;
168
169 swap_info.total_amount_out += simulation_response.return_amount;
170 swap_info.total_amount_in += payment.amount;
```

BVSS:

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

Recommendation:

It is recommended to have separate variables to track the swapped amounts.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commit 22e954a.

4.23 (HAL-23) UNBOND WAIT LISTS CAN BE MIGRATED EVEN IF THE CONTRACT IS NOT PAUSED - LOW (3.4)

Description:

The migrate_unbond_wait_lists function in the **krp-staking-contracts/basset_sei_hub** contract allows migrating the unbond wait lists without previously verifying whether the contract is paused or not, which could break the initial assumption about the current state of the contract and would arise unexpected situations, e.g.: reverting unbonding requests.

Code Location:

The migrate_unbond_wait_lists function assumes that the contract is pause without verifying it:

Listing 23: krp-staking-contracts/contracts/basset_sei_hub/src/state.rs (Lines 303-309)

```
265 pub fn migrate_unbond_wait_lists(
266  storage: &mut dyn Storage,
267  limit: Option<u32>,
268 ) -> StdResult<Response> {
269  let (removed_keys, num_migrated_entries) = {
270   let old_unbond_wait_list_entries = read_old_unbond_wait_lists(
 ↳ storage, limit)?;
271   if old_unbond_wait_list_entries.is_empty() {
272    return Ok(Response::new().add_attributes(vec![
273     attr("action", "migrate_unbond_wait_lists"),
274     attr("num_migrated_entries", "0"),
275    ]));
276   }
277
278   let mut num_migrated_entries: u32 = 0;
279   let mut new_unbond_wait_list: Bucket<UnbondWaitEntity> =
280    Bucket::multilevel(storage, &[NEW_PREFIX_WAIT_MAP]);
```

```
281   let mut removed_keys: Vec<Vec<u8>> = vec![];
282
283   for res in old_unbond_wait_list_entries {
284    let (key, amount) = res?;
285    let unbond_wait_entity = UnbondWaitEntity {
286     bsei_amount: amount,
287     stsei_amount: Uint128::zero(),
288    };
289    new_unbond_wait_list.save(&key, &unbond_wait_entity)?;
290    removed_keys.push(key);
291    num_migrated_entries += 1;
292   }
293
294   (removed_keys, num_migrated_entries)
295  };
296
297  let mut old_unbond_wait_list: Bucket<Uint128> =
298  Bucket::multilevel(storage, &[OLD_PREFIX_WAIT_MAP]);
299  for key in removed_keys {
300   old_unbond_wait_list.remove(&key);
301  }
302
303  // unpause contract if we've migrated all unbond wait lists
304  let old_unbond_wait_list_entries = read_old_unbond_wait_lists(
 ↳ storage, Some(1u32))?;
305  if old_unbond_wait_list_entries.is_empty() {
306   let mut params: Parameters = PARAMETERS.load(storage)?;
307   params.paused = Some(false);
308   PARAMETERS.save(storage, &params)?;
309  }
310
311  Ok(Response::new().add_attributes(vec![
312   attr("action", "migrate_unbond_wait_lists"),
313   attr("num_migrated_entries", num_migrated_entries.to_string()),
314  ]))
315 }
```

BVSS:

AO:A/AC:L/AX:M/C:N/I:M/A:N/D:N/Y:N/R:N/S:U (3.4)

Recommendation:

Update the logic of the migrate_unbond_wait_lists function to verify if the contract is paused before further execution.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commit 37cb49a.

4.24 (HAL-24) UNCHECKED MAX LOAN-TO-VALUE RATIO - LOW (3.4)

Description:

The whitelist_collateral function in the **krp-cdp-contracts/central_control** contract does not verify that max_ltv parameter is lower than 1. If it is mistakenly set to a value greater than 1, users will be able to redeem more coins than the value of their deposited collaterals.

This issue also applies to the register_whitelist and update_whitelist functions in the **krp-market-contracts/overseer** contract.

Code Location:

The whitelist_collateral function in the **krp-cdp-contracts/central_control** contract:

Listing 24: krp-cdp-contracts/contracts/central_control/src/contract.rs (Line 884)

```
889 pub fn whitelist_collateral(
890     deps: DepsMut,
891     info: MessageInfo,
892     name: String,
893     symbol: String,
894     max_ltv: Decimal256,
895     custody_contract: CanonicalAddr,
896     collateral_contract: CanonicalAddr,
897     reward_book_contract: CanonicalAddr,
898 ) -> Result<Response, ContractError> {
899     let config = read_config(deps.storage)?;
900
901     if deps.api.addr_canonicalize(info.sender.as_str())? != config
 ↳ .owner_addr {
902         return Err(ContractError::Unauthorized(
903             "whitelist_collateral".to_string(),
904             info.sender.to_string(),
905         ));
906     }
```

```
907
908     if max_ltv >= Decimal256::one() {
909         return Err(ContractError::MaxLtvExceedsLimit {});
910     }
911
912     let data = WhitelistElem {
913         name,
914         symbol,
915         max_ltv,
916         custody_contract,
917         collateral_contract: collateral_contract.clone(),
918         reward_book_contract,
919     };
920     store_whitelist_elem(deps.storage, &collateral_contract, &data
 ↳ )?;
921     Ok(Response::default())
922 }
```

BVSS:

AO:A/AC:L/AX:M/C:N/I:N/A:N/D:N/Y:M/R:N/S:U (3.4)

Recommendation:

It is recommended to check the max_ltv parameter each time it is modified from an external input and verify that its value is less than 1.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commits 13e9a4f and 8f2be6a.

4.25 (HAL-25) FUNCTIONALITY TO UPDATE FEES IS NOT APPLIED CORRECTLY - LOW (3.4)

Description:

The assert_fees function in update_config from the **krp-cdp-contracts/liquidation_queue** and **krp-market-contracts/liquidation_queue** contracts is not applied correctly. As a consequence, some valid values will not be accepted, see example below:

Current values:

- bid_fee = 0.3
- liquidator_fee = 0.6

New values:

- bid_fee = 0.5
- liquidator_fee = 0.4

The new values will not be accepted despite that **0.5** + **0.4** < **1**. The assert_fees function should be applied after trying to change both values.

Code Location:

Snippet of update_config function in the **krp-market-contracts/liquidation_queue** contract:

Listing 25: krp-market-contracts/contracts/liquidation_queue/src/contract.rs (Lines 205-213)

```
175 pub fn update_config(
176     deps: DepsMut,
177     info: MessageInfo,
```

```
178     owner: Option<String>,
179     oracle_contract: Option<String>,
180     safe_ratio: Option<Decimal256>,
181     bid_fee: Option<Decimal256>,
182     liquidator_fee: Option<Decimal256>,
183     liquidation_threshold: Option<Uint256>,
184     price_timeframe: Option<u64>,
185     waiting_period: Option<u64>,
186     overseer: Option<String>,
187 ) -> Result<Response, ContractError> {
188     let mut config: Config = read_config(deps.storage)?;
189     if deps.api.addr_canonicalize(info.sender.as_str())? != config
 ↳ .owner {
190         return Err(ContractError::Unauthorized {});
191     }
192
193     if let Some(owner) = owner {
194         config.owner = deps.api.addr_canonicalize(&owner)?;
195     }
196
197     if let Some(oracle_contract) = oracle_contract {
198         config.oracle_contract = deps.api.addr_canonicalize(&
 ↳ oracle_contract)?;
199     }
200
201     if let Some(safe_ratio) = safe_ratio {
202         config.safe_ratio = safe_ratio;
203     }
204
205     if let Some(bid_fee) = bid_fee {
206         assert_fees(bid_fee + config.liquidator_fee)?;
207         config.bid_fee = bid_fee;
208     }
209
210     if let Some(liquidator_fee) = liquidator_fee {
211         assert_fees(liquidator_fee + config.bid_fee)?;
212         config.liquidator_fee = liquidator_fee;
213     }
```

BVSS:

AO:A/AC:L/AX:M/C:N/I:N/A:N/D:N/Y:M/R:N/S:U (3.4)

Recommendation:

It is recommended to apply the assert_fees function after trying to change both values (bid_fee and liquidator_fee).

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commits 5530267 and da2a2a3.

4.26 (HAL-26) ARBITRARY MESSAGES CAN BE EXECUTED ON BEHALF OF THE HUB - LOW (3.4)

Description:

The execute_update_global function in the **krp-staking-contracts/basset_sei_hub** contract allows any user to execute arbitrary messages (airdrop_hooks) in the **airdrop_registry_contract** contract (out-of-scope for this assessment) on behalf of **basset_sei_hub**, which could arise unexpected situations in the protocol, like unauthorized changes.

Code Location:

Listing 26: krp-staking-contracts/contracts/basset_sei_hub/src/contract.rs (Lines 335-341)

```
329 if airdrop_hooks.is_some() {
330  let registry_addr =
331   deps.api
332   .addr_humanize(&config.airdrop_registry_contract.ok_or_else(|| {
333    StdError::generic_err("the airdrop registry contract must have
 ↳ been registered")
334   })?)?;
335  for msg in airdrop_hooks.unwrap() {
336   messages.push(CosmosMsg::Wasm(WasmMsg::Execute {
337    contract_addr: registry_addr.to_string(),
338    msg,
339    funds: vec![],
340   }))
341  }
342 }
```

BVSS:

AO:A/AC:L/AX:M/C:N/I:M/A:N/D:N/Y:N/R:N/S:U (3.4)

Recommendation:

Add an access control in the execute_update_global function to ensure that only the **creator** address can execute it.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commit 31b6d03.

4.27 (HAL-27) UNRELIABLE SOURCE OF RANDOMNESS - LOW (3.3)

Description:

The get_winning function in the **krp-token-contracts/treasure** contract uses the following parameters as a source of randomness to determine the winning numbers when pre-minting NFTs:

- Block time
- Block height
- Position of the transaction in the block

Although those parameters are volatile and make it harder to guess the winning numbers, they are not a reliable source of randomness from a security perspective.

Code Location:

Listing 27: krp-token-contracts/contracts/treasure/src/handler.rs (Lines 365-371)

```
358 let mut win_nft_num = 0u64;
359 let mut lost_nft_num = 0u64;
360 let record_id = generate_next_global_id(deps.storage)?;
361 let winning_num = &config.winning_num;
362 let mod_num = &config.mod_num;
363 for i in 0..mint_num {
364  let unique_factor = record_id + i;
365  let winning = get_winning(
366   env.clone(),
367   unique_factor.to_string(),
368   vec![],
369   winning_num,
370   mod_num,
371  )?;
372  if winning {
373   win_nft_num += 1;
```

```
374  } else {
375   lost_nft_num += 1;
376  }
377 }
```

BVSS:

AO:A/AC:H/AX:L/C:N/I:N/A:N/D:N/Y:C/R:N/S:U (3.3)

Recommendation:

It is recommended to use an oracle that provides a safe and secure entropy source, e.g.: Nois.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commit 9d12155.

4.28 (HAL-28) WITHDRAWAL COULD GET STUCK IF THERE ARE NO MORE UNBONDINGS - LOW (3.1)

Description:

If users unbond before the epoch period using the execute_unbond or execute_unbond_stsei functions in the **krp-staking-contracts/basset_sei_hub** contract, they won't be able to withdraw their tokens later unless someone else unbonds after the epoch period and triggers the undelegation. As a consequence, in an edge scenario, the withdrawal could get stuck.

Code Location:

Relevant code fragments from the execute_unbond and execute_unbond_stsei functions:

Listing 28: krp-staking-contracts/contracts/basset_sei_hub/src/unbond.rs

```
84 // If the epoch period is passed, the undelegate message would be
↳ sent.
85 if passed_time > epoch_period {
86  let mut undelegate_msgs =
87   process_undelegations(&mut deps, env, &mut current_batch, &mut
↳ state)?;
88  messages.append(&mut undelegate_msgs);
89 }
```

Listing 29: krp-staking-contracts/contracts/basset_sei_hub/src/unbond.rs

```
450 // If the epoch period is passed, the undelegate message would be
↳ sent.
451 if passed_time > epoch_period {
452  let mut undelegate_msgs =
453   process_undelegations(&mut deps, env, &mut current_batch, &mut
↳ state)?;
```

```
454  messages.append(&mut undelegate_msgs);
455 }
```

BVSS:

AO:A/AC:L/AX:M/C:N/I:N/A:H/D:H/Y:N/R:P/S:U (3.1)

Recommendation:

It is recommended to enable a mechanism that allows users to trigger the undelegation once the epoch period has finished, even if there are no more unbondings.

Remediation Plan:

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

4.29 (HAL-29) OWNERSHIP CAN BE TRANSFERRED WITHOUT CONFIRMATION - LOW (2.5)

Description:

An incorrect use of the execute_update_config, update_config, change_owner, change_gov or update_staking_config functions in some contracts can set owner to an invalid address and inadvertently lose control of them, which cannot be undone in any way.

The affected contracts are the following:

- krp-staking-contracts/basset_sei_hub
- krp-staking-contracts/basset_sei_rewards_dispatcher
- krp-staking-contracts/basset_sei_validators_registry
- krp-cdp-contracts/central_control
- krp-cdp-contracts/custody
- krp-cdp-contracts/liquidation_queue
- krp-cdp-contracts/reward_book
- krp-cdp-contracts/stable_pool
- krp-market-contracts/custody_base
- krp-market-contracts/custody_bsei
- krp-market-contracts/distribution_model
- krp-market-contracts/interest_model
- krp-market-contracts/liquidation_queue
- krp-market-contracts/market
- krp-market-contracts/overseer
- krp-oracle/oracle_pyth
- krp-token-contracts/boost
- krp-token-contracts/dispatcher
- krp-token-contracts/distribute
- krp-token-contracts/fund
- krp-token-contracts/keeper
- krp-token-contracts/seilor

- krp-token-contracts/staking
- krp-token-contracts/ve_seilor
- swap-extension/swap_sparrow

Code Location:

Example - Code of the update_config function in the **krp-market-contracts/custody_base** contract:

Listing 30: krp-market-contracts/contracts/custody_base/src/contract.rs (Line 136)

```
123 pub fn update_config(
124     deps: DepsMut,
125     info: MessageInfo,
126     owner: Option<Addr>,
127     liquidation_contract: Option<Addr>,
128 ) -> Result<Response, ContractError> {
129     let mut config: Config = read_config(deps.storage)?;
130 
131     if deps.api.addr_canonicalize(info.sender.as_str())? != config
 ↳ .owner {
132         return Err(ContractError::Unauthorized {});
133     }
134 
135     if let Some(owner) = owner {
136         config.owner = deps.api.addr_canonicalize(owner.as_str())
 ↳ ?;
137     }
138 
139     if let Some(liquidation_contract) = liquidation_contract {
140         config.liquidation_contract = deps.api.addr_canonicalize(
 ↳ liquidation_contract.as_str())?;
141     }
142 
143     store_config(deps.storage, &config)?;
144     Ok(Response::new().add_attributes(vec![attr("action", "
 ↳ update_config")]))
145 }
```

BVSS:

AO:A/AC:L/AX:H/C:N/I:N/A:H/D:N/Y:N/R:N/S:U (2.5)

Recommendation:

It is recommended to split ownership transfer functionality into `set_owner` and `accept_ownership` functions. The latter function allows the transfer to be completed by the recipient.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commits `2a7389f`, `93247af`, `a1d8a9c`, `f3c96c8`, `5ab2069`, `2407815` and `6dec8f1`.

4.30 (HAL-30) COMMENTED TRANSACTION MESSAGE - LOW (2.5)

Description:

The claim_rewards function in the **krp-market-contracts/market** contract could be called by any user in order to get back the rewards generated by their deposits. Currently, this operation will not work since the message which contains the information on the transaction is commented, so a blank message will be executed.

Code Location:

Code fragment of the claim_rewards function in the **krp-market-contracts/market** contract:

Listing 31: krp-market-contracts/contracts/market/src/borrow.rs (Lines 232-250)

```
226     let claim_amount = liability.pending_rewards * Uint256::one();
227     liability.pending_rewards = liability.pending_rewards -
 ↳ Decimal256::from_uint256(claim_amount);
228
229     store_state(deps.storage, &state)?;
230     store_borrower_info(deps.storage, &borrower_raw, &liability)?;
231     let messages: Vec<CosmosMsg> = vec![];
232     // let messages: Vec<CosmosMsg> = if !claim_amount.is_zero() {
233     //     vec![CosmosMsg::Wasm(WasmMsg::Execute {
234     //         contract_addr: deps
235     //             .api
236     //             .addr_humanize(&config.distributor_contract)?
237     //             .to_string(),
238     //         funds: vec![],
239     //         msg: to_binary(&FaucetExecuteMsg::Spend {
240     //             recipient: if let Some(to) = to {
241     //                 to.to_string()
242     //             } else {
243     //                 borrower.to_string()
244     //             },
245     //             amount: claim_amount.into(),
```

```
246     //          })?,
247     //      })]
248     // } else {
249     //      vec![]
250     // };
251
252     Ok(Response::new().add_messages(messages).add_attributes(vec![
253         attr("action", "claim_rewards"),
254         attr("claim_amount", claim_amount),
255     ]))
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:L/R:N/S:U (2.5)

Recommendation:

It is recommended to remove the comment from the transaction information to execute the rewards transfer.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commits e766b7c.

4.31 (HAL-31) UNCHECKED REDEEM FEE - LOW (2.1)

Description:

The instantiate and update_config functions in the **krp-cdp-contracts/central_control** contract do not verify that redeem_fee is lower than 1. If it is mistakenly set to a value greater than 1, the operation of redeeming stable coins will always panic because of an underflow error.

Code Location:

The instantiate and update_config functions do not verify that redeem_fee is lower than 1:

Listing 32: krp-cdp-contracts/contracts/central_control/src/contract.rs (Line 58)

```
44 #[cfg_attr(not(feature = "library"), entry_point)]
45 pub fn instantiate(
46  deps: DepsMut,
47  _env: Env,
48  _info: MessageInfo,
49  msg: InstantiateMsg,
50 ) -> Result<Response, ContractError> {
51  let api = deps.api;
52  let config = Config {
53   owner_addr: api.addr_canonicalize(&msg.owner_addr.as_str())?,
54   oracle_contract: api.addr_canonicalize(&msg.oracle_contract.
↳ as_str())?,
55   pool_contract: api.addr_canonicalize(&msg.pool_contract.as_str()
↳ )?,
56   liquidation_contract: api.addr_canonicalize(&msg.
↳ liquidation_contract.as_str())?,
57   epoch_period: msg.epoch_period,
58   redeem_fee: msg.redeem_fee,
59   stable_denom: msg.stable_denom,
60  };
61  store_config(deps.storage, &config)?;
```

```
62
63  Ok(Response::default())
64 }
```

Listing 33: krp-cdp-contracts/contracts/central_control/src/contract.rs (Lines 502-504)

```
498 if let Some(epoch_period) = epoch_period {
499  config.epoch_period = epoch_period;
500 }
501
502 if let Some(redeem_fee) = redeem_fee {
503  config.redeem_fee = redeem_fee;
504 }
505
506 store_config(deps.storage, &config)?;
507 Ok(Response::default())
```

BVSS:

AO:A/AC:L/AX:M/C:N/I:N/A:M/D:M/Y:N/R:P/S:U (2.1)

Recommendation:

It is recommended to verify that redeem_fee is lower than 1 in the functions mentioned above.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commit 54b01dd.

4.32 (HAL-32) UNCHECKED KEEPER RATE - LOW (2.1)

Description:

The instantiate and execute_update_config functions in the **krp-staking-contracts/basset_sei_rewards_dispatcher** contract do not verify that krp_keeper_rate is lower than 1. If it is mistakenly set to a value greater than 1, the operation of rewards dispatching will always panic because of an underflow error.

Code Location:

The instantiate and execute_update_config functions do not verify that krp_keeper_rate is lower than 1:

Listing 34: krp-staking-contracts/contracts/basset_sei_rewards_dispatcher/src/contract.rs (Line 45)

```
31 #[cfg_attr(not(feature = "library"), entry_point)]
32 pub fn instantiate(
33  deps: DepsMut,
34  _env: Env,
35  info: MessageInfo,
36  msg: InstantiateMsg,
37 ) -> StdResult<Response> {
38  let conf = Config {
39   owner: deps.api.addr_canonicalize(info.sender.as_str())?,
40   hub_contract: deps.api.addr_canonicalize(&msg.hub_contract)?,
41   bsei_reward_contract: deps.api.addr_canonicalize(&msg.
↳ bsei_reward_contract)?,
42   bsei_reward_denom: msg.bsei_reward_denom,
43   stsei_reward_denom: msg.stsei_reward_denom,
44   krp_keeper_address: deps.api.addr_canonicalize(&msg.
↳ krp_keeper_address)?,
45   krp_keeper_rate: msg.krp_keeper_rate,
46   swap_contract: deps.api.addr_canonicalize(&msg.swap_contract)?,
47   swap_denoms: msg.swap_denoms,
48   oracle_contract: deps.api.addr_canonicalize(&msg.oracle_contract
↳ )?,
```

```
 49  };
 50
 51  store_config(deps.storage, &conf)?;
 52  Ok(Response::default())
 53 }
```

Listing 35: krp-staking-contracts/contracts/basset_sei_rewards_dispatcher/src/contract.rs (Lines 160-165)

```
153 if let Some(_b) = bsei_reward_denom {
154  CONFIG.update(deps.storage, |mut last_config| -> StdResult<_> {
155   last_config.bsei_reward_denom = _b;
156   Ok(last_config)
157  })?;
158 }
159
160 if let Some(r) = krp_keeper_rate {
161  CONFIG.update(deps.storage, |mut last_config| -> StdResult<_> {
162   last_config.krp_keeper_rate = r;
163   Ok(last_config)
164  })?;
165 }
```

BVSS:

AO:A/AC:L/AX:M/C:N/I:N/A:M/D:N/Y:M/R:P/S:U (2.1)

Recommendation:

It is recommended to verify that krp_keeper_rate is lower than 1 in the functions mentioned above.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commit c20f645.

4.33 (HAL-33) MAXIMUM AMOUNT OF TOKENS TO MINT IS NOT VALIDATED - LOW (2.1)

Description:

The update_config function in the **krp-token-contracts/ve_seilor** contract does not verify that max_minted should be greater than total_minted. If this parameter is mistakenly set (i.e.: lower than total_minted), it will be possible to mint new **ve_seilor** tokens.

Code Location:

The update_config function does not verify that max_minted should be greater than total_minted:

Listing 36: krp-token-contracts/contracts/ve_seilor/src/handler.rs (Lines 16-19)

```
 8 pub fn update_config(deps: DepsMut, info: MessageInfo, max_minted:
↳  Option<Uint128>, fund: Option<Addr>, gov: Option<Addr>) -> Result
↳ <Response, ContractError> {
 9  let mut vote_config = read_vote_config(deps.storage)?;
10
11  if info.sender != vote_config.gov {
12   return Err(ContractError::Unauthorized {});
13  }
14
15  let mut attrs = vec![attr("action", "update_config"), attr("
↳ sender", info.sender.to_string())];
16  if let Some(max_minted) = max_minted {
17   vote_config.max_minted = max_minted.clone();
18   attrs.push(attr("max_minted", max_minted.to_string()));
19  }
```

BVSS:

AO:A/AC:L/AX:M/C:N/I:M/A:N/D:M/Y:N/R:P/S:U (2.1)

Recommendation:

It is recommended to verify that max_minted is greater than total_minted in the function mentioned above.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commit 6b82713.

4.34 (HAL-34) UNCHECKED PARAMETERS IN DISPATCHER CONTRACT - LOW (2.1)

Description:

The instantiate or update_config functions in the **krp-token-contracts/dispatcher** contract do not verify the following parameters:

- start_lock_period_time >= **current block time**
- duration_per_period > 0
- periods > 0

If one of those parameters is mistakenly set, it could arise in unexpected situations, e.g.: panicking when users try to claim.

Code Location:

The instantiate or update_config functions do not verify the appropriate boundaries:

Listing 37: krp-token-contracts/contracts/dispatcher/src/contract.rs (Lines 30,31,32)

```
17 #[cfg_attr(not(feature = "library"), entry_point)]
18 pub fn instantiate(
19  deps: DepsMut,
20  _env: Env,
21  info: MessageInfo,
22  msg: InstantiateMsg,
23 ) -> StdResult<Response> {
24  let gov = msg.gov.unwrap_or_else(|| info.sender.clone());
25
26  let global_config = GlobalConfig {
27   gov,
28   claim_token: msg.claim_token,
29   total_lock_amount: msg.total_lock_amount,
30   start_lock_period_time: msg.start_lock_period_time,
31   duration_per_period: msg.duration_per_period,
32   periods: msg.periods,
```

```
 };

 let global_state = GlobalState {
  total_user_lock_amount: Uint256::zero(),
  total_user_claimed_lock_amount: Uint256::zero(),
 };

 set_contract_version(deps.storage, CONTRACT_NAME,
↳ CONTRACT_VERSION)?;
 store_global_config(deps.storage, &global_config)?;
 store_global_state(deps.storage, &global_state)?;

 Ok(Response::new().add_attribute("action", "instantiate"))
}
```

Listing 38: krp-token-contracts/contracts/dispatcher/src/handler.rs (Lines 50-60)

```
if let Some(start_lock_period_time) = msg.start_lock_period_time {
 // check current block time > start_lock_period_time
 if config.start_lock_period_time < env.block.time.seconds() {
  return Err(ContractError::InvalidStartLockPeriodTime {});
 }
 config.start_lock_period_time = start_lock_period_time.clone();
 attrs.push(attr(
  "start_lock_period_time",
  start_lock_period_time.to_string(),
 ));
}

store_global_config(deps.storage, &config)?;

Ok(Response::default().add_attributes(attrs))
```

BVSS:

AO:A/AC:L/AX:M/C:N/I:N/A:M/D:N/Y:M/R:P/S:U (2.1)

Recommendation:

It is recommended to enforce the appropriate boundaries for the parameters mentioned above.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commits 183833f, 8e08417 and a48bf9e.

4.35 (HAL-35) UNCHECKED PARAMETERS IN TREASURE CONTRACT - LOW (2.1)

Description:

The instantiate or update_config functions in the **krp-token-contracts/treasure** contract do not verify the following parameters:

- start_lock_time >= **current block time**
- end_lock_time > start_lock_time
- nft_start_pre_mint_time >= **current block time**
- nft_start_pre_mint_time > end_lock_time
- nft_end_pre_mint_time > nft_start_pre_mint_time

If one of those parameters is mistakenly set, it could arise unexpected situations, e.g.: users couldn't withdraw rewards or pre-mint NFTs.

Code Location:

The instantiate or update_config functions do not verify the appropriate boundaries:

Listing 39: krp-token-contracts/contracts/treasure/src/contract.rs (Lines 30,31,37,38)

```
17 #[cfg_attr(not(feature = "library"), entry_point)]
18 pub fn instantiate(
19  deps: DepsMut,
20  _env: Env,
21  info: MessageInfo,
22  msg: InstantiateMsg,
23 ) -> StdResult<Response> {
24  let sender = info.clone().sender;
25  let gov = msg.gov.unwrap_or(sender.clone());
26
27  let config = TreasureConfig {
28   gov: gov.clone(),
29   lock_token: msg.lock_token.clone(),
```

```
30   start_lock_time: msg.start_lock_time,
31   end_lock_time: msg.end_lock_time,
32   dust_reward_per_second: msg.dust_reward_per_second,
33   withdraw_delay_duration: msg.withdraw_delay_duration,
34   winning_num: msg.winning_num,
35   mod_num: msg.mod_num,
36   punish_receiver: msg.punish_receiver,
37   nft_start_pre_mint_time: msg.nft_start_pre_mint_time,
38   nft_end_pre_mint_time: msg.nft_end_pre_mint_time,
39   no_delay_punish_coefficient: msg.no_delay_punish_coefficient,
40   mint_nft_cost_dust: msg.mint_nft_cost_dust,
41  };
```

Listing 40: krp-token-contracts/contracts/treasure/src/handler.rs (Lines 37-40,41-44)

```
32 if let Some(lock_token) = config_msg.lock_token {
33  deps.api.addr_validate(lock_token.clone().as_str())?;
34  config.lock_token = lock_token.clone();
35  attrs.push(attr("lock_token", lock_token.to_string()));
36 }
37 if let Some(start_lock_time) = config_msg.start_lock_time {
38  config.start_lock_time = start_lock_time.clone();
39  attrs.push(attr("start_lock_time", start_lock_time.to_string()));
40 }
41 if let Some(end_lock_time) = config_msg.end_lock_time {
42  config.end_lock_time = end_lock_time.clone();
43  attrs.push(attr("end_lock_time", end_lock_time.to_string()));
44 }
```

Listing 41: krp-token-contracts/contracts/treasure/src/handler.rs (Lines 72-78,79-85)

```
66 if let Some(punish_receiver) = config_msg.punish_receiver {
67  deps.api.addr_validate(punish_receiver.clone().as_str())?;
68  config.punish_receiver = punish_receiver.clone();
69  attrs.push(attr("punish_receiver", punish_receiver.to_string()));
70 }
71
72 if let Some(nft_start_pre_mint_time) = config_msg.
↳ nft_start_pre_mint_time {
73  config.nft_start_pre_mint_time = nft_start_pre_mint_time.clone();
```

```
74  attrs.push(attr(
75   "nft_start_pre_mint_time",
76   nft_start_pre_mint_time.to_string(),
77  ));
78 }
79 if let Some(nft_end_pre_mint_time) = config_msg.
↳ nft_end_pre_mint_time {
80  config.nft_end_pre_mint_time = nft_end_pre_mint_time.clone();
81  attrs.push(attr(
82   "nft_end_pre_mint_time",
83   nft_end_pre_mint_time.to_string(),
84  ));
85 }
```

BVSS:

AO:A/AC:L/AX:M/C:N/I:N/A:M/D:M/Y:N/R:P/S:U (2.1)

Recommendation:

It is recommended to enforce the appropriate boundaries for the parameters mentioned above.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commit 8776456.

4.36 (HAL-36) DURATION IS NOT VALIDATED IN STAKING CONTRACT - LOW (2.1)

Description:

The instantiate and update_staking_duration functions in **krp-token-contracts/staking** contract do not validate that the duration is greater than 0. If it is mistakenly set to **0**, the operation of notifying reward amount will always panic because of a division by 0.

Code Location:

The instantiate and update_staking_duration functions do not verify that duration is greater than 0:

Listing 42: krp-token-contracts/contracts/staking/src/constract.rs (Line 46)

```
23 #[cfg_attr(not(feature = "library"), entry_point)]
24 pub fn instantiate(
25  deps: DepsMut,
26  _env: Env,
27  info: MessageInfo,
28  msg: InstantiateMsg,
29 ) -> Result<Response, ContractError> {
30  let gov = msg.gov.unwrap_or_else(|| info.sender.clone());
31
32  set_contract_version(deps.storage, CONTRACT_NAME,
↳ CONTRACT_VERSION)?;
33
34  let staking_config = StakingConfig {
35   gov,
36   staking_token: msg.staking_token,
37   rewards_token: msg.rewards_token,
38   boost: msg.boost,
39   fund: msg.fund,
40   reward_controller_addr: msg.reward_controller_addr,
41  };
```

```
42
43  store_staking_config(deps.storage, &staking_config)?;
44
45  let staking_state = StakingState {
46   duration: msg.duration,
47   finish_at: Uint128::zero(),
48   updated_at: Uint128::zero(),
49   reward_rate: Uint256::zero(),
50   reward_per_token_stored: Uint128::zero(),
51   total_supply: Uint128::zero(),
52  };
```

Listing 43: krp-token-contracts/contracts/staking/src/handler.rs (Line 93)

```
75 pub fn update_staking_duration(
76  deps: DepsMut,
77  env: Env,
78  info: MessageInfo,
79  duration: Uint128,
80 ) -> Result<Response, ContractError> {
81  let staking_config = read_staking_config(deps.storage)?;
82  let mut staking_state = read_staking_state(deps.storage)?;
83  if info.sender.ne(&staking_config.gov) {
84   return Err(ContractError::Unauthorized {});
85  }
86
87  let current_time = Uint128::from(env.block.time.seconds());
88  if staking_state.finish_at > current_time {
89   return Err(ContractError::Std(StdError::generic_err(
90    "duration can only be updated after the end of the current
↳ period",
91   )));
92  }
93  staking_state.duration = duration.clone();
94
95  store_staking_state(deps.storage, &staking_state)?; // update
↳ state
```

BVSS:

AO:A/AC:L/AX:M/C:N/I:N/A:M/D:N/Y:M/R:P/S:U (2.1)

Recommendation:

It is recommended to verify that duration is greater than 0 in the functions mentioned above.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commit 9def1db.

4.37 (HAL-37) CLAIMABLE TIME IS NOT VALIDATED IN FUND CONTRACT - LOW (2.1)

Description:

The instantiate and update_fund_config functions in the **krp-token-contracts/fund** contract do not validate that the claim_able_time is greater than the current block time. If it is mistakenly set (i.e.: lower than current block time), users won't be able to unstake tokens.

Code Location:

The instantiate and update_fund_config functions do not verify that claim_able_time is greater than the current block time:

Listing 44: krp-token-contracts/contracts/fund/src/contract.rs (Line 44)

```
23 #[cfg_attr(not(feature = "library"), entry_point)]
24 pub fn instantiate(
25  deps: DepsMut,
26  _env: Env,
27  info: MessageInfo,
28  msg: InstantiateMsg,
29 ) -> StdResult<Response> {
30  set_contract_version(deps.storage, CONTRACT_NAME,
↳ CONTRACT_VERSION)?;
31
32  let gov = msg.gov.unwrap_or_else(|| info.sender.clone());
33
34  let config = FundConfig {
35   gov,
36   ve_seilor_addr: msg.ve_seilor_addr,
37   seilor_addr: msg.seilor_addr,
38   kusd_denom: msg.kusd_denom,
39   kusd_reward_addr: msg.kusd_reward_addr,
40   kusd_reward_total_amount: Uint128::zero(),
41   kusd_reward_total_paid_amount: Uint128::zero(),
```

```
42   reward_per_token_stored: Uint128::zero(),
43   exit_cycle: msg.exit_cycle,
44   claim_able_time: msg.claim_able_time,
45  };
```

Listing 45: krp-token-contracts/contracts/fund/src/handler.rs (Lines 62-65)

```
62 if let Some(claim_able_time) = msg.claim_able_time {
63  config.claim_able_time = claim_able_time.clone();
64  attrs.push(attr("claim_able_time", claim_able_time.to_string()));
65 }
66 store_fund_config(deps.storage, &config)?;
67 Ok(Response::new().add_attributes(attrs))
```

BVSS:

AO:A/AC:L/AX:M/C:N/I:N/A:M/D:M/Y:N/R:P/S:U (2.1)

Recommendation:

It is recommended to verify that claim_able_time is greater than the current block time in the functions mentioned above.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commit d17a174.

4.38 (HAL-38) EXCHANGE RATE COULD INCREASE INDEFINITELY - LOW (2.1)

Description:

The values of bsei_exchange_rate and stsei_exchange_rate can increase indefinitely (i.e.: greater than 1) by directly burning **bsei** and **stsei** tokens, respectively. Although this issue is not immediately exploitable, if those exchange rates get increased too much eventually, it could create some overflow situations in the protocol when bonding, unbonding, withdrawing or converting.

Code Location:

The execute_burn function in the **basset_sei_token_bsei** and **basset_sei_token_stsei** contracts is not adequately restricted:

Listing 46: krp-staking-contracts/contracts/basset_sei_token_bsei/src/handler.rs (Line 79)

```
69 pub fn execute_burn(
70  deps: DepsMut,
71  env: Env,
72  info: MessageInfo,
73  amount: Uint128,
74 ) -> Result<Response, ContractError> {
75  let sender = info.sender.clone();
76  let reward_contract = query_reward_contract(&deps)?;
77  let hub_contract = deps.api.addr_humanize(&read_hub_contract(deps
↳ .storage)?)?;
78
79  let res: Response = cw20_burn(deps, env, info, amount)?;
```

Listing 47: krp-staking-contracts/contracts/basset_sei_token_stsei/src/handler.rs (Line 63)

```
43 pub fn execute_burn(
44  deps: DepsMut,
```

```
45  env: Env,
46  info: MessageInfo,
47  amount: Uint128,
48 ) -> Result<Response, ContractError> {
49  let hub_contract = deps.api.addr_humanize(&HUB_CONTRACT.load(deps
↳ .storage)?)?;
50
51  let mut messages = vec![SubMsg::new(CosmosMsg::Wasm(WasmMsg::
↳ Execute {
52   contract_addr: hub_contract.to_string(),
53   msg: to_binary(&CheckSlashing {})?,
54   funds: vec![],
55  }))];
56  if info.sender != hub_contract {
57   messages.push(SubMsg::new(CosmosMsg::Wasm(WasmMsg::Execute {
58    contract_addr: hub_contract.to_string(),
59    msg: to_binary(&CheckSlashing {})?,
60    funds: vec![],
61   })))
62  }
63  let res = cw20_burn(deps, env, info, amount)?;
```

BVSS:

AO:A/AC:L/AX:H/C:N/I:M/A:M/D:N/Y:N/R:N/S:U (2.1)

Recommendation:

It is recommended to restrict that only **basset_sei_hub** contract burns the tokens.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commit 67cc445.

4.39 (HAL-39) PAIR KEY CAN CONTAIN DUPLICATED ASSETS - LOW (2.1)

Description:

The update_pair_config function in the **swap-extension/swap_sparrow** contract does not verify that asset_infos contains different assets, which could allow that the owner mistakenly registers an invalid pair of assets in the contract.

Code Location:

The update_pair_config function does not verify that asset_infos contains different assets:

Listing 48: swap-extension/contracts/swap_sparrow/src/handler.rs

```
#[allow(clippy::too_many_arguments)]
pub fn update_pair_config(deps: DepsMut, info: MessageInfo,
 asset_infos: [AssetInfo; 2],
 pair_address: Addr, max_spread: Option<Decimal>,
 to: Option<Addr>) -> Result<Response, ContractError> {
 let config = read_config(deps.storage)?;
 if info.sender != config.owner {
  return Err(ContractError::Unauthorized {});
 }

 let mut pair_config = PairConfig {
  pair_address: pair_address.clone(),
  is_disabled: false,
  max_spread: None,
  to: None,
 };

 if let Some(max_spread) = max_spread {
  pair_config.max_spread = Some(max_spread);
 }
 if let Some(to) = to {
  pair_config.to = Some(to);
 }
```

```
35
36  let pair_key = pair_key(&asset_infos);
37  store_pair_configs(deps.storage, &pair_key, &pair_config)?;
38
39  Ok(Response::new().add_attributes(vec![
40   ("action", "update_pair_config"),
41   ("pair_address", pair_address.as_str()),
42   ("max_spread", max_spread.unwrap_or_default().to_string().as_str
↳ ()), ]))
43 }
```

BVSS:

AO:A/AC:L/AX:M/C:N/I:L/A:N/D:L/Y:N/R:N/S:U (2.1)

Recommendation:

It is recommended to update the logic of update_pair_config function, in such a way that it verifies that asset_infos contains different assets.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commit 3ea0429.

4.40 (HAL-40) ASSETS COULD MISMATCH WITH THE ONES IN PAIR ADDRESS - LOW (2.1)

Description:

The update_pair_config function in **swap_sparrow** contract does not verify that the assets in pair_address are the same ones as in asset_infos, which could allow that the owner mistakenly registers an invalid pair of assets in the contract.

Code Location:

The update_pair_config function does not verify that the assets in pair_address are the same ones as in asset_infos:

Listing 49: swap-extension/contracts/swap_sparrow/src/handler.rs

```
#[allow(clippy::too_many_arguments)]
pub fn update_pair_config(deps: DepsMut, info: MessageInfo,
 asset_infos: [AssetInfo; 2],
 pair_address: Addr, max_spread: Option<Decimal>,
 to: Option<Addr>) -> Result<Response, ContractError> {
 let config = read_config(deps.storage)?;
 if info.sender != config.owner {
  return Err(ContractError::Unauthorized {});
 }

 let mut pair_config = PairConfig {
  pair_address: pair_address.clone(),
  is_disabled: false,
  max_spread: None,
  to: None,
 };

 if let Some(max_spread) = max_spread {
  pair_config.max_spread = Some(max_spread);
 }
 if let Some(to) = to {
```

```
33   pair_config.to = Some(to);
34  }
35
36  let pair_key = pair_key(&asset_infos);
37  store_pair_configs(deps.storage, &pair_key, &pair_config)?;
38
39  Ok(Response::new().add_attributes(vec![
40   ("action", "update_pair_config"),
41   ("pair_address", pair_address.as_str()),
42   ("max_spread", max_spread.unwrap_or_default().to_string().as_str
↳ ()), ]))
43 }
```

BVSS:

AO:A/AC:L/AX:M/C:N/I:L/A:N/D:L/Y:N/R:N/S:U (2.1)

Recommendation:

It is recommended to update the logic of update_pair_config function, in such a way that it verifies that the assets in pair_address are the same ones as in asset_infos.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commit aa6fd58.

4.41 (HAL-41) IMMUTABLE VARIABLES CAN BE CHANGED IN STAKING CONTRACT - INFORMATIONAL (1.7)

Description:

The update_staking_config function in the **krp-token-contracts/staking** contract allows updating the values of staking_token and rewards_token variables, which should be immutable. As a consequence, some unexpected situations could arise in the protocol, e.g.: users that can't withdraw their staked tokens.

Code Location:

Listing 50: krp-token-contracts/contracts/staking/src/handler.rs (Lines 39,35)

```
37 if let Some(staking_token) = update_struct.staking_token {
38  deps.api.addr_validate(staking_token.clone().as_str())?; //
↳ validate staking token address
39  staking_config.staking_token = staking_token.clone();
40  attrs.push(attr("staking_token", staking_token.to_string()));
41 }
42
43 if let Some(rewards_token) = update_struct.rewards_token {
44  deps.api.addr_validate(rewards_token.clone().as_str())?; //
↳ validate rewards token address
45  staking_config.rewards_token = rewards_token.clone();
46  attrs.push(attr("rewards_token", rewards_token.to_string()));
47 }
```

BVSS:

AO:A/AC:L/AX:M/C:N/I:M/A:N/D:N/Y:N/R:P/S:U (1.7)

Recommendation:

It is recommended to remove the possibility to update the values of the `staking_token` and `rewards_token` variables.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commit `c8afb00`.

4.42 (HAL-42) MARKETING INFO IS NOT VALIDATED AT INSTANTIATION - INFORMATIONAL (1.7)

Description:

If marketing info is not included at instantiation time, the execute_update_marketing and execute_upload_logo functions will always return an Unauthorized error message because marketing is **None**. The affected contracts are the following:

- **krp-staking-contracts/basset_sei_token_stsei**
- **krp-token-contracts/seilor**
- **krp-token-contracts/ve_seilor**

Code Location:

The marketing parameter is optional when instantiating the **krp-staking-contracts/basset_sei_token_stsei** contract:

Listing 51: krp-staking-contracts/contracts/basset_sei_token_stsei/src/msg.rs (Line 27)

```
20 #[derive(Serialize, Deserialize, JsonSchema)]
21 pub struct TokenInitMsg {
22  pub name: String,
23  pub symbol: String,
24  pub decimals: u8,
25  pub initial_balances: Vec<Cw20Coin>,
26  pub hub_contract: String,
27  pub marketing: Option<InstantiateMarketingInfo>,
28 }
```

The marketing parameter is optional when instantiating the **krp-token-contracts/seilor** contract:

Listing 52: krp-token-contracts/contracts/seilor/src/contract.rs (Lines 46-53)

```
46 if let Some(marketing) = cw20_instantiate_msg.marketing {
47  cw20_instantiate_msg.marketing = Some(InstantiateMarketingInfo {
48   project: marketing.project,
49   description: marketing.description,
50   logo: marketing.logo,
51   marketing: Some(gov.to_string()),
52  });
53 }
54
55 let ins_res = cw20_instantiate(deps.branch(), env, info,
↳ cw20_instantiate_msg);
56 if let Err(err) = ins_res {
57  return Err(ContractError::Std(StdError::generic_err(err.to_string
↳ ())));
58 }
```

The marketing parameter is optional when instantiating the **krp-token-contracts/ve_seilor** contract:

Listing 53: krp-token-contracts/contracts/ve_seilor/src/contract.rs (Lines 56-58)

```
49 cw20_instantiate_msg.marketing = if let Some(marketing) =
↳ cw20_instantiate_msg.marketing {
50  Some(InstantiateMarketingInfo {
51   project: marketing.project,
52   description: marketing.description,
53   logo: marketing.logo,
54   marketing: Option::from(gov.clone().to_string()),
55  })
56 } else {
57  None
58 };
```

BVSS:

AO:A/AC:L/AX:M/C:N/I:N/A:L/D:N/Y:N/R:N/S:U (1.7)

Recommendation:

It is recommended to ensure that marketing and marketing.marketing are different from **None** when instantiating the contracts listed above.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commits fb5272f and 060cbb5.

4.43 (HAL-43) LOCK AMOUNT IN GLOBAL STATE IS NOT VALIDATED WHEN CLAIMING - INFORMATIONAL (1.7)

Description:

The user_claim function in the **krp-token-contracts/dispatcher** contract does not validate that global_state.total_user_claimed_lock_amount is less or equal than global_state.total_user_lock_amount. This situation could generate a panic during the claiming operation under edge scenarios.

Code Location:

The user_claim function only validates that claimed_lock_amount <= total_user_lock_amount for the user state, but not for the global state:

Listing 54: krp-token-contracts/contracts/dispatcher/src/handler.rs (Lines 187-189)

```
182 // check claimable amount is not zero
183 if claimable_amount == Uint256::zero() {
184  return Err(ContractError::UserClaimAmountIsZero(sender.clone()));
185 }
186
187 if user_state.claimed_lock_amount > user_state.
 ↳ total_user_lock_amount {
188  return Err(ContractError::UserClaimLockAmountTooLarge(sender.
 ↳ clone()));
189 }
190
191 store_user_state(deps.storage, &sender, &user_state)?;
192 store_global_state(deps.storage, &global_state)?;
```

BVSS:

AO:A/AC:L/AX:H/C:N/I:M/A:N/D:N/Y:N/R:N/S:U (1.7)

Recommendation:

It is recommended to validate that `total_user_claimed_lock_amount` <= `total_user_lock_amount` for the global state.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commit `6ce1c7e`.

4.44 (HAL-44) UNCHECKED VALIDATOR ADDRESSES - INFORMATIONAL (1.2)

Description:

The validator addresses in the **krp-token-contracts/basset_sei_validators_registry** contract are not validated by addr_validate before saving them to storage in the instantiate and add_validator functions. In edge scenarios, this situation would allow storing invalid addresses, which could generate that some operations are reverted.

Code Location:

The instantiate function does not use addr_validate before saving the validator addresses to storage:

Listing 55: krp-staking-contracts/contracts/basset_sei_validators_registry/src/contract.rs (Lines 45-47)

```
31 pub fn instantiate(
32  deps: DepsMut,
33  _env: Env,
34  info: MessageInfo,
35  msg: InstantiateMsg,
36 ) -> StdResult<Response> {
37  CONFIG.save(
38   deps.storage,
39   &Config {
40    owner: deps.api.addr_canonicalize(info.sender.as_str())?,
41    hub_contract: deps.api.addr_canonicalize(msg.hub_contract.
↳ as_str())?,
42   },
43  )?;
44
45  for v in msg.registry {
46   REGISTRY.save(deps.storage, v.address.as_str().as_bytes(), &v)?;
47  }
48
49  Ok(Response::default())
```

The add_validator function does not use addr_validate before saving the validator addresses to storage:

Listing 56: krp-staking-contracts/contracts/basset_sei_validators_registry/src/contract.rs (Lines 114-118)

```
101 pub fn add_validator(
102  deps: DepsMut,
103  _env: Env,
104  info: MessageInfo,
105  validator: Validator,
106 ) -> StdResult<Response> {
107  let config = CONFIG.load(deps.storage)?;
108  let owner_address = deps.api.addr_humanize(&config.owner)?;
109  let hub_address = deps.api.addr_humanize(&config.hub_contract)?;
110  if info.sender != owner_address && info.sender != hub_address {
111   return Err(StdError::generic_err("unauthorized"));
112  }
113
114  REGISTRY.save(
115   deps.storage,
116   validator.address.as_str().as_bytes(),
117   &validator,
118  )?;
119  Ok(Response::default())
120 }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:L/A:N/D:N/Y:N/R:P/S:U (1.2)

Recommendation:

It is recommended to validate addresses using the addr_validate function before saving them to the storage.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commit f3c1794.

4.45 (HAL-45) UNCHECKED SAFE RATIO - INFORMATIONAL (0.8)

Description:

The instantiate and update_config functions from the **krp-cdp-contracts/liquidation_queue** and **krp-market-contracts/liquidation_queue** contracts do not verify that safe_ratio is lower than 1.

If it is mistakenly set to a value greater than 1, some unexpected results could appear or even a panic when querying about the liquidation amount.

Code Location:

Snippet of update_config function in the **krp-market-contracts/liquidation_queue** contract:

Listing 57: krp-market-contracts/contracts/liquidation_queue/src/contract.rs (Lines 180,201-203)

```
175 pub fn update_config(
176     deps: DepsMut,
177     info: MessageInfo,
178     owner: Option<String>,
179     oracle_contract: Option<String>,
180     safe_ratio: Option<Decimal256>,
181     bid_fee: Option<Decimal256>,
182     liquidator_fee: Option<Decimal256>,
183     liquidation_threshold: Option<Uint256>,
184     price_timeframe: Option<u64>,
185     waiting_period: Option<u64>,
186     overseer: Option<String>,
187 ) -> Result<Response, ContractError> {
188     let mut config: Config = read_config(deps.storage)?;
189     if deps.api.addr_canonicalize(info.sender.as_str())? != config
 ↳ .owner {
190         return Err(ContractError::Unauthorized {});
191     }
192
193     if let Some(owner) = owner {
```

```
194             config.owner = deps.api.addr_canonicalize(&owner)?;
195         }
196
197         if let Some(oracle_contract) = oracle_contract {
198             config.oracle_contract = deps.api.addr_canonicalize(&
 ↳ oracle_contract)?;
199         }
200
201         if let Some(safe_ratio) = safe_ratio {
202             config.safe_ratio = safe_ratio;
203         }
```

BVSS:

AO:A/AC:L/AX:M/C:N/I:N/A:L/D:N/Y:N/R:P/S:U (0.8)

Recommendation:

It is recommended to verify that the safe_ratio parameter is lower than 1 in the functions mentioned above.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commits a373e5b, 9ec168e and d225015.

4.46 (HAL-46) REDUNDANT LOGIC - INFORMATIONAL (0.0)

Description:

The execute_convert_to_basset and the execute_convert_to_native functions in the **krp-basset-convert/krp_basset_converter** contract contain redundant logic, which could mean a possible error inside the logical conditions or, more likely, that the code is redundant and should be simplified as part of the DRY (Don't Repeat Yourself) principle used as a best practice in software development to improve the maintainability of code during all phases of its lifecycle.

Code Location:

Redundant logic in the execute_convert_to_basset function:

Listing 58: krp-basset-convert/contracts/krp_basset_converter/src/contract.rs (Line 107)

```
100 pub(crate) fn execute_convert_to_basset(
101  deps: DepsMut,
102  _env: Env,
103  info: MessageInfo,
104 ) -> StdResult<Response> {
105  let config = read_config(deps.storage)?;
106
107  if config.native_denom.is_none() || config.native_denom.is_none()
 ↳ {
108   return Err(StdError::generic_err(
109    "native denom must be registered first",
110   ));
111  }
```

Redundant logic in the execute_convert_to_native function:

Listing 59: krp-basset-convert/contracts/krp_basset_converter/src/contract.rs (Line 162)

```
154 pub(crate) fn execute_convert_to_native(
155  deps: DepsMut,
156  _env: Env,
157  _info: MessageInfo,
158  amount: Uint128,
159  sender: String,
160 ) -> StdResult<Response> {
161  let config = read_config(deps.storage)?;
162  if config.native_denom.is_none() || config.native_denom.is_none()
 ↳ {
163   return Err(StdError::generic_err(
164    "native or basset token must be registered first",
165   ));
166  }
```

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 simplify the code mentioned above to avoid cases where redundant logic appears.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commit 2bd0553.

4.47 (HAL-47) REPEATED EXECUTION MESSAGES - INFORMATIONAL (0.0)

Description:

The ExecuteMsg::MigrateUnbondWaitList and ExecuteMsg::UpdateParams messages are repeated in the execute function from **krp-staking-contracts/basset_sei_hub** contract. Although this situation doesn't pose a security risk, it's included in the report as part of the DRY (Don't Repeat Yourself) principle used as a best practice in software development to improve the maintainability of code during all phases of its lifecycle.

Code Location:

The ExecuteMsg::MigrateUnbondWaitList and ExecuteMsg::UpdateParams messages are repeated in the execute function:

Listing 60: krp-staking-contracts/contracts/basset_sei_hub/src/contract.rs (Lines 110-112,114-132)

```
108 #[cfg_attr(not(feature = "library"), entry_point)]
109 pub fn execute(deps: DepsMut, env: Env, info: MessageInfo, msg:
 ↳ ExecuteMsg) -> StdResult<Response> {
110  if let ExecuteMsg::MigrateUnbondWaitList { limit } = msg {
111   return migrate_unbond_wait_lists(deps.storage, limit);
112  }
113
114  if let ExecuteMsg::UpdateParams {
115   epoch_period,
116   unbonding_period,
117   peg_recovery_fee,
118   er_threshold,
119   paused,
120  } = msg
121  {
122   return execute_update_params(
123    deps,
124    env,
```

```
125     info,
126     epoch_period,
127     unbonding_period,
128     peg_recovery_fee,
129     er_threshold,
130     paused,
131   );
132 }
```

Listing 61: krp-staking-contracts/contracts/basset_sei_hub/src/contract.rs (Lines 149-164)

```
147 ExecuteMsg::WithdrawUnbonded {} => execute_withdraw_unbonded(deps,
 ↳  env, info),
148 ExecuteMsg::CheckSlashing {} => execute_slashing(deps, env),
149 ExecuteMsg::UpdateParams {
150  epoch_period,
151  unbonding_period,
152  peg_recovery_fee,
153  er_threshold,
154  paused,
155 } => execute_update_params(
156   deps,
157   env,
158   info,
159   epoch_period,
160   unbonding_period,
161   peg_recovery_fee,
162   er_threshold,
163   paused,
164  ),
```

Listing 62: krp-staking-contracts/contracts/basset_sei_hub/src/contract.rs (Line 217)

```
213 ExecuteMsg::RedelegateProxy {
214  src_validator,
215  redelegations,
216 } => execute_redelegate_proxy(deps, env, info, src_validator,
 ↳ redelegations),
217 ExecuteMsg::MigrateUnbondWaitList { limit: _ } => Err(StdError::
 ↳ generic_err("forbidden")),
```

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 update the codebase to call only one version of the duplicated messages and remove the other ones to avoid potential mistakes.

Remediation Plan:

ACKNOWLEDGED: The Kryptonite team acknowledged this finding.

4.48 (HAL-48) STAKING TOKEN CALLS CHECK SLASHING TWICE - INFORMATIONAL (0.0)

Description:

The ExecuteMsg::Burn message sent to the **krp-staking-contracts/basset_sei_token_stsei** contract triggers CheckSlashing twice on the **hub**, if the sender of that message is not the **hub** contract. The second call is redundant and does simply recompute the exchange rates another time, which leads to computational and gas waste.

Code Location:

Listing 63: krp-staking-contracts/contracts/basset_sei_token_stsei/src/handler.rs (Line 51)

```
51 let mut messages = vec![SubMsg::new(CosmosMsg::Wasm(WasmMsg::
↳ Execute {
52     contract_addr: hub_contract.to_string(),
53     msg: to_binary(&CheckSlashing {})?,
54     funds: vec![],
55 }))];
56 if info.sender != hub_contract {
57     messages.push(SubMsg::new(CosmosMsg::Wasm(WasmMsg::Execute {
58         contract_addr: hub_contract.to_string(),
59         msg: to_binary(&CheckSlashing {})?,
60         funds: vec![],
61     })))
62 }
```

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 review the wanted behavior and remove the unnecessary duplicated code.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commit e7c4805.

4.49 (HAL-49) UNIMPLEMENTED MESSAGE - INFORMATIONAL (0.0)

Description:

The QueryMsg::GetBufferedRewards message is not implemented in the **krp-staking-contracts/basset_sei_rewards_dispatcher** contract and could generate a panic if it is called. Although it is a minor issue, it is advisable to fix the logic in a **production environment** contract.

Code Location:

Listing 64: krp-staking-contracts/contracts/basset_sei_rewards_dispatcher/src/contract.rs (Line 515)

```
511 #[cfg_attr(not(feature = "library"), entry_point)]
512 pub fn query(deps: Deps, _env: Env, msg: QueryMsg) -> StdResult<
 ↳ Binary> {
513  match msg {
514   QueryMsg::Config {} => to_binary(&query_config(deps)?),
515   QueryMsg::GetBufferedRewards {} => unimplemented!(),
516  }
517 }
```

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 unused message or implement its logic if necessary.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commit 3debcf9.

4.50 (HAL-50) UNUSED MESSAGES - INFORMATIONAL (0.0)

Description:

Along the codebase, there are some messages that are unused. This situation is not security-related, but mentioned in the report as part of the best practices in software development to improve the readability of code during all phases of its lifecycle. The affected messages are the following:

- **krp-cdp-contracts/liquidation_queue:** ExecuteMsg::ExecuteBid
- **krp-token-contracts/ve_seilor:** QueryMsg::GetPastTotalSupply

Code Location:

The ExecuteMsg::ExecuteBid message is not used:

Listing 65: krp-cdp-contracts/contracts/liquidation_queue/src/contract.rs

```
116 ExecuteMsg::ExecuteBid {
117  liquidator,
118  repay_address,
119  fee_address,
120  collateral_denom,
121  amount,
122 } => {
123  let sender = deps
124     .api
125     .addr_canonicalize(&info.sender.as_str())?
126     .to_string();
127  let collateral_token = collateral_denom;
128  execute_liquidation(
129   deps,
130   env,
131   sender,
132   liquidator,
133   repay_address,
```

```
134   fee_address,
135   collateral_token,
136   amount,
137  )
138 }
```

The QueryMsg::GetPastTotalSupply message is not used:

Listing 66: krp-token-contracts/contracts/ve_seilor/src/contract.rs

```
148 QueryMsg::GetPastTotalSupply { block_number } => {
149  to_binary(&get_past_total_supply(deps, env, block_number)?)
150 }
```

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 unused messages in the code to make it more readable.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commits 80f3137 and dd0a194.

4.51 (HAL-51) UNUSED VARIABLES - INFORMATIONAL (0.0)

Description:

Along the codebase, there are some variables that are unused. This situation is not security-related, but mentioned in the report as part of the best practices in software development to improve the readability of code during all phases of its lifecycle. The affected variables are the following:

- **krp-token-contracts/distribute:** RuleConfig.lock_end_time
- **krp-token-contracts/treasure:** TreasureState.total_withdraw_amount and TreasureState.total_unlock_amount

Code Location:

The lock_end_time variable is not used:

Listing 67: krp-token-contracts/contracts/distribute/src/state.rs (Line 21)

```
14 #[derive(Serialize, Deserialize, Clone, Debug, PartialEq,
↳ JsonSchema)]
15 pub struct RuleConfig {
16  pub rule_name: String,
17  pub rule_owner: Addr,
18  pub rule_total_amount: u128,
19  pub start_release_amount: u128,
20  pub lock_start_time: u64,
21  pub lock_end_time: u64,
22  pub start_linear_release_time: u64,
23  pub end_linear_release_time: u64,
24  pub unlock_linear_release_amount: u128,
25  pub unlock_linear_release_time: u64,
26  pub linear_release_per_second: u128,
27 }
```

The total_unlock_amount and total_withdraw_amount variables are not used:

Listing 68: krp-token-contracts/contracts/treasure/src/state.rs (Lines 39,40)

```
33 #[derive(Serialize, Deserialize, Clone, Debug, PartialEq,
↳ JsonSchema)]
34 pub struct TreasureState {
35  pub current_unlock_amount: Uint128,
36  pub current_locked_amount: Uint128,
37
38  pub total_locked_amount: Uint128,
39  pub total_unlock_amount: Uint128,
40  pub total_withdraw_amount: Uint128,
41  pub total_punish_amount: Uint128,
42  pub total_cost_dust_amount: Uint128,
43
44  pub total_win_nft_num: u64,
45  pub total_lose_nft_num: u64,
46 }
```

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 unused variables in the code to make it more readable.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commit 0e7162a.

4.52 (HAL-52) USELESS FUNCTIONS - INFORMATIONAL (0.0)

Description:

The deduct_tax and deduct_tax_new functions in **krp-market-contracts/moneymarket/packages** do not perform any kind of tax deduction or operation over the input amount. These functions can be removed.

Code Location:

Code of the deduct_tax and deduct_tax_new functions in the **Moneymarket** package:

Listing 69: krp-market-contracts/packages/moneymarket/src/querier.rs

```
126 pub fn deduct_tax(_deps: Deps, coin: Coin) -> StdResult<Coin> {
127     //let tax_amount = compute_tax(deps, &coin)?;
128     Ok(Coin {
129         denom: coin.denom,
130         //amount: (Uint256::from(coin.amount) - tax_amount).into()
 ↳ ,
131         amount: Uint256::from(coin.amount).into(),
132     })
133 }
134
135 pub fn deduct_tax_new(_deps: Deps, burn_amount: Uint128) ->
 ↳ StdResult<Uint256> {
136     // let tax_cap = Uint256::one();
137     // let protocal_fee_rate = Decimal256::from_ratio(5, 1000);
138
139     // Ok(std::cmp::min(
140     //     Uint256::from(burn_amount) * Decimal256::one() -
 ↳ Uint256::from(burn_amount) / (Decimal256::one() +
 ↳ protocal_fee_rate),
141     //     tax_cap,
142     //     ))
143     Ok(Uint256::from(burn_amount))
144 }
```

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 modify the functions if they are really going to be used to deduct some taxes, or to remove them.

Remediation Plan:

ACKNOWLEDGED: The Kryptonite team acknowledged this finding.

4.53 (HAL-53) HARDCODED DENOM - INFORMATIONAL (0.0)

Description:

The fund_reserves function in the **krp-market-contracts/overseer** contract uses the hardcoded uusd denom to filter incoming funds.

The rest of the contract always uses the config.stable_denom parameter as a reference to the stable coin stored in the contract.

Code Location:

Code of the fund_reserve function in the **Overseer** contract:

Listing 70: contracts/overseer/src/contract.rs (Line 653)

```
652 pub fn fund_reserve(deps: DepsMut, info: MessageInfo) -> Result<
 ↳ Response, ContractError> {
653     let sent_uusd = match info.funds.iter().find(|x| x.denom == "
 ↳ uusd") {
654         Some(coin) => coin.amount,
655         None => Uint128::zero(),
656     };
657 
658     let mut overseer_epoch_state: EpochState = read_epoch_state(
 ↳ deps.storage)?;
659     overseer_epoch_state.prev_interest_buffer += Uint256::from(
 ↳ sent_uusd);
660     store_epoch_state(deps.storage, &overseer_epoch_state)?;
661 
662     let mut dyn_rate_state: DynrateState = read_dynrate_state(deps
 ↳ .storage)?;
663     dyn_rate_state.prev_yield_reserve += Decimal256::from_ratio(
 ↳ Uint256::from(sent_uusd), 1);
664     store_dynrate_state(deps.storage, &dyn_rate_state)?;
665 
666     Ok(Response::new().add_attributes(vec![
667         attr("action", "fund_reserve"),
668         attr("funded_amount", sent_uusd.to_string()),
```

```
669     ]))
670 }
```

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 replace the hardcoded `uusd` denom with the `config.stable_denom` parameter in case the stable currency for funding the contract needs to be changed.

Remediation Plan:

SOLVED: The Kryptonite team solved this issue in commit `ba0efdb`.

THANK YOU FOR CHOOSING

// HALBORN