Dexter - Dexter Core

CosmWasm Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **August 22nd, 2022 - February 10th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	08/22/2022	Lukasz Mikula
0.2	Document Update	10/29/2022	Lukasz Mikula
0.3	Document Update	11/03/2022	Lukasz Mikula
0.4	Document Update	11/08/2022	Lukasz Mikula
0.5	Draft Version	11/11/2022	Jakub Heba
0.6	Draft Review	11/14/2022	Gabi Urrutia
0.7	Document Update	12/01/2022	Lukasz Mikula
0.8	Document Update Review	12/07/2022	Gabi Urrutia
1.0	Remediation Plan	01/09/2023	Emiliano Carmona
1.1	Remediation Plan Review	01/09/2023	Luis Quispe Gonzales
1.2	Remediation Plan Review	01/09/2023	Gabi Urrutia
2.0	Document Update	02/10/2023	Elena Maranon
2.1	Document Update Review	02/15/2023	Luis Quispe Gonzales
2.2	Document Update Review	02/16/2023	Gabi Urrutia
2.3	Remediation Plan	02/24/2023	Elena Maranon
2.4	Remediation Plan Review	02/24/2023	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Luis Quispe Gonzales	Halborn	Luis.QuispeGonzales@halborn.com
Lukasz Mikula	Halborn	Lukasz.Mikula@halborn.com
Jakub Heba	Halborn	Jakub.Heba@halborn.com
Emiliano Carmona	Halborn	Emiliano.Carmona@halborn.com
Elena Maranon	Halborn	Elena.Maranon@halborn.com
Gonzalo Junquera	Halborn	Gonzalo.Junquera@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Dexter engaged Halborn to conduct a security audit on their smart contracts beginning on August 22nd, 2022 and ending on February 10th, 2023. The security assessment was scoped to the smart contracts provided in the GitHub repository dexter-core, commit hashes and further details can be found in the Scope section of this report.

The Dexter project is a decentralized exchange that allows the creation of pools for the exchange of tokens, adding liquidity to them and staking the LP tokens obtained in return to earn some rewards.

1.2 AUDIT SUMMARY

The team at Halborn assigned a two full-time security engineers to audit the security of the smart contract. The security engineers are blockchain and smart-contract security experts with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

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

In summary, Halborn identified some improvements to reduce the likelihood and impact of risks, which were partially addressed by Dexter team. The main ones are the following:

- Validate newly created pools, so they are not the same pairs and type as already existing ones.
- Ensure users are protected against slippage even if they do not specify maximal slippage parameters.
- Handle cases when more than one type of native assets are sent along with the multi-swap request.

- Modify the access control of the update_total_fee_bps function to allow to be called from the vault.
- Modify the update_config function to not overwrite by mistake some of the configuration parameters.
- Implement the logic of 'GiveOut' swaps via the router or disable its query simulation.
- Increase the validation over the LP tokens.
- Increase the XML validation.
- Add an ownership transfer mechanism to keeper contract.
- Add an additional state to enable the emergency status.
- Avoid saving useless empty objects with wrong information in the storage.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the Rust code and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose.
- Smart contract manual code review and walkthrough.
- Manual testing by custom scripts and fuzzers.
- Scanning of Rust files for vulnerabilities, security hotspots or bugs.
- Static Analysis of security for scoped contract, and imported functions.
- Testnet deployment.

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.

1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

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

1.4 SCOPE

First round of testing (August 22nd - November 10th):

1. CosmWasm Smart Contracts
   (a) Repository: dexter-core
   (b) Commit ID: 1d810ec and 5b3819d
   (c) Contracts in scope:
      i. generator_proxy
      ii. ref_staking
      iii. vesting
      iv. keeper
      v. lp_token
      vi. vault

Second round of testing (November 7th - November 10th):

1. CosmWasm Smart Contracts
   (a) Repository: dexter-core
   (b) Commit ID: 1d810ec
   (c) Contracts in scope:
      i. stable_5pool
      ii. stable_pool
      iii. weighted_pool
      iv. xyk_pool

Third round of testing (November 21st - December 1st):

1. CosmWasm Smart Contracts
   (a) Repository: dexter-core
   (b) Commit ID: 091313f
   (c) Contracts in scope:
      i. router

Fourth round of testing (Dec 5th - Jan 20th):

1. CosmWasm Smart Contracts
   (a) Repository: dexter-core
   (b) Commit ID: 6b49e0f
   (c) Contracts in scope:
      i. xyk_pool
      ii. stable_pool
      iii. stable_5pool
      iv. weighted_pool
      v. ref_staking
      vi. multi_staking
      vii. lp_token
      viii. generator
      ix. generator_proxy
      x. keeper
      xi. router
      xii. vesting
      xiii. vault

Fifth round of testing (Jan 20th - Feb 2nd): New commit with fixes and changes

1. CosmWasm Smart Contracts
   (a) Repository: dexter-core
   (b) Commit ID: c45b9dc
   (c) Contracts in scope:
      i. xyk_pool
      ii. stable_pool
      iii. stable_5pool
      iv. weighted_pool
      v. ref_staking
      vi. multi_staking
      vii. lp_token
      viii. generator
      ix. generator_proxy
      x. keeper
      xi. router
      xii. vesting
      xiii. vault

Sixth round of testing (Feb 2nd - Feb 10th): New commit with fixes

1. CosmWasm Smart Contracts
   (a) Repository: dexter-core
   (b) Commit ID: 06dfa23
   (c) Contracts in scope:
      i. stable_5pool
      ii. weighted_pool
      iii. multi_staking
      iv. lp_Token

v. keeper

vi. router

vii. vault

Final remediations (Feb 24th): New commit with fixes

1. CosmWasm Smart Contracts

    (a) Repository: dexter-core

    (b) Commit ID: 6773bfb

    (c) Contracts in scope:

    i. stable_5pool

    ii. weighted_pool

    iii. multi_staking

    iv. lp_Token

    v. keeper

    vi. router

    vii. vault

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

EXECUTIVE OVERVIEW

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	1	4	6	15

LIKELIHOOD

IMPACT

🟨	🟨	🟥	🟥	🟪
🟩	🟨	🟨	🟥 (HAL-01)	🟥
🟩 (HAL-07) (HAL-08)	🟩	🟨 (HAL-02) (HAL-03) (HAL-04) (HAL-05)	🟨	🟥
⬜ (HAL-12) (HAL-13) (HAL-14) (HAL-15) (HAL-16) (HAL-17) (HAL-18)	🟩 (HAL-09) (HAL-10) (HAL-11)	🟩	🟨	🟨
⬜ (HAL-19) (HAL-20) (HAL-21) (HAL-22) (HAL-23) (HAL-24) (HAL-25) (HAL-26)	⬜	🟩	🟩 (HAL-06)	🟨

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) REPEATED POOLS MAY BE CREATED	High	RISK ACCEPTED
(HAL-02) MAXIMAL SLIPPAGE IS NOT ENFORCED	Medium	RISK ACCEPTED
(HAL-03) FUNDS WILL BE LOST IF ADDITIONAL NATIVE TOKENS ARE SENT	Medium	RISK ACCEPTED
(HAL-04) INADEQUATE ACCESS CONTROL PREVENTS UPDATING THE POOL FEE	Medium	SOLVED - 01/31/2023
(HAL-05) SOME PARAMETERS MAY BE OVERWRITTEN WHEN UPDATING THE VAULT CONFIGURATION	Medium	SOLVED - 01/19/2023
(HAL-06) GIVEOUT SWAP TYPE CANNOT BE EXECUTED	Low	NOT APPLICABLE
(HAL-07) INSUFFICIENT LP TOKEN VALIDATION	Low	SOLVED - 12/01/2022
(HAL-08) INSUFFICIENT XML VALIDATION	Low	RISK ACCEPTED
(HAL-09) LACK OF TRANSFER OWNERSHIP MECHANISM IN KEEPER CONTRACT	Low	SOLVED - 11/30/2022
(HAL-10) EMERGENCY UNSTAKE DO NOT CHECKS FOR EMERGENCY STATUS	Low	SOLVED - 02/02/2023
(HAL-11) IMPROPER ERROR HANDLING (MULTISTAKING)	Low	SOLVED - 01/31/2023
(HAL-12) EMPTY MIGRATE FUNCTION	Informational	ACKNOWLEDGED
(HAL-13) LACK OF INPUT VALIDATION (MULTISTAKING)	Informational	SOLVED - 01/13/2023
(HAL-14) WRONG DATA INPUT	Informational	SOLVED - 12/14/2022
(HAL-15) NON-ZERO CHECK PERFORMED AFTER TRANSACTION	Informational	SOLVED - 01/31/2023
(HAL-16) LACK OF ADDRESS VALIDATION	Informational	SOLVED - 02/21/2023

(HAL-17) WRONG IMPLEMENTATION CAN WASTE SOME GAS (HELPER)	Informational	SOLVED - 01/31/2023
(HAL-18) EARLIER VALIDATION CAN SAVE SOME GAS (VAULT)	Informational	SOLVED - 01/31/2023
(HAL-19) INCOMPLETE MIGRATION	Informational	SOLVED - 12/01/2022
(HAL-20) OVERFLOW-CHECKS BEST PRACTICES	Informational	ACKNOWLEDGED
(HAL-21) USELESS CODE	Informational	SOLVED - 01/31/2023
(HAL-22) UNNECESSARY PARAMETERS	Informational	SOLVED - 02/02/2023
(HAL-23) USE OF MAGIC NUMBERS	Informational	SOLVED - 02/24/2023
(HAL-24) UNCHECKED MATH	Informational	SOLVED - 01/31/2023
(HAL-25) IMPROPER ERROR HANDLING (POOLS)	Informational	SOLVED - 02/02/2023
(HAL-26) STORAGE CLEAN UP	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) REPEATED POOLS MAY BE CREATED - HIGH

Description:

It was observed that the CreatePoolInstance function allows for creating repeated pools. As a result, the liquidity in such pools might be low, which will negatively impact the resulting price of underlying assets. Since the aim of having a liquidity pool is to be competitive, the liquidity should be possibly maximal to offer most competitive prices to the users, which in turn encourages users to use the liquidity pools. If certain coin pairs will be split over numerous pools, this will be difficult to achieve.

For instance, there are already examples of existing protocols that disallow same pair registration such as Astroport AMM or Aptos Pontem Network AMM.

Code Location:

Below code shows a sample unit test that allows for creation of repeated pools:

Listing 1: contracts/vault/tests/integration.rs (Lines 527-529,531-534,536-538)

```rust
    #[test]
fn test_create_pool_instance() {
    let mut app = mock_app();
    let owner = String::from("owner");
    let token_code_id = store_token_code(&mut app);
    let vault_instance = instantiate_contract(&mut app, &Addr::unchecked(owner.clone()));

    // Create Token X
    let init_msg = TokenInstantiateMsg {
        name: "x_token".to_string(),
        symbol: "X-Tok".to_string(),
        decimals: 18,
        initial_balances: vec![],
        mint: Some(MinterResponse {
            minter: owner.to_string(),
            cap: None,
        }),
        marketing: None,
    };
    let token_instance0 = app
        .instantiate_contract(
            token_code_id,
            Addr::unchecked(owner.clone()),
            &init_msg,
            &[],
            "x_token",
            None,
        )
        .unwrap();

    // Create Token Y
    let init_msg = TokenInstantiateMsg {
        name: "y_token".to_string(),
        symbol: "Y-Tok".to_string(),
        decimals: 18,
        initial_balances: vec![],
        mint: Some(MinterResponse {
            minter: owner.to_string(),
            cap: None,
        }),
        marketing: None,
    };
    let token_instance1 = app
        .instantiate_contract(
            token_code_id,
            Addr::unchecked(owner.clone()),
            &init_msg,
            &[],
            "y_token",
            None,
        )
        .unwrap();

    let asset_infos = vec![
        AssetInfo::Token {
            contract_addr: token_instance0.clone(),
        },
        AssetInfo::Token {
            contract_addr: token_instance1.clone(),
        },
    ];

    let msg = ExecuteMsg::CreatePoolInstance {
        pool_type: PoolType::Xyk {},
        asset_infos: asset_infos.to_vec(),
        init_params: None,
        lp_token_name: None,
        lp_token_symbol: None,
    };


    let res = app
        .execute_contract(Addr::unchecked(owner), vault_instance.clone(), &msg, &[])
        .unwrap();

    let owner2 = String::from("owner");
        let res2 = app
        .execute_contract(Addr::unchecked(owner2), vault_instance.clone(), &msg, &[])
        .unwrap();

        assert_eq!(res.events[1].attributes[1], res2.events[1].attributes[1]);
        assert_eq!(res.events[1].attributes[2], res2.events[1].attributes[2]);
        assert_eq!(res.events[1].attributes[2], attr("pool_type", "xyk"));

}
```

Risk Level:

Likelihood - 4

Impact - 4

Recommendation:

It is recommended to include a validity check in function CreatePoolInstance in order to ensure if a pool with the same pool_type and asset_infos already exists.

Remediation Plan:

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

3.2 (HAL-02) MAXIMAL SLIPPAGE IS NOT ENFORCED - MEDIUM

Description:

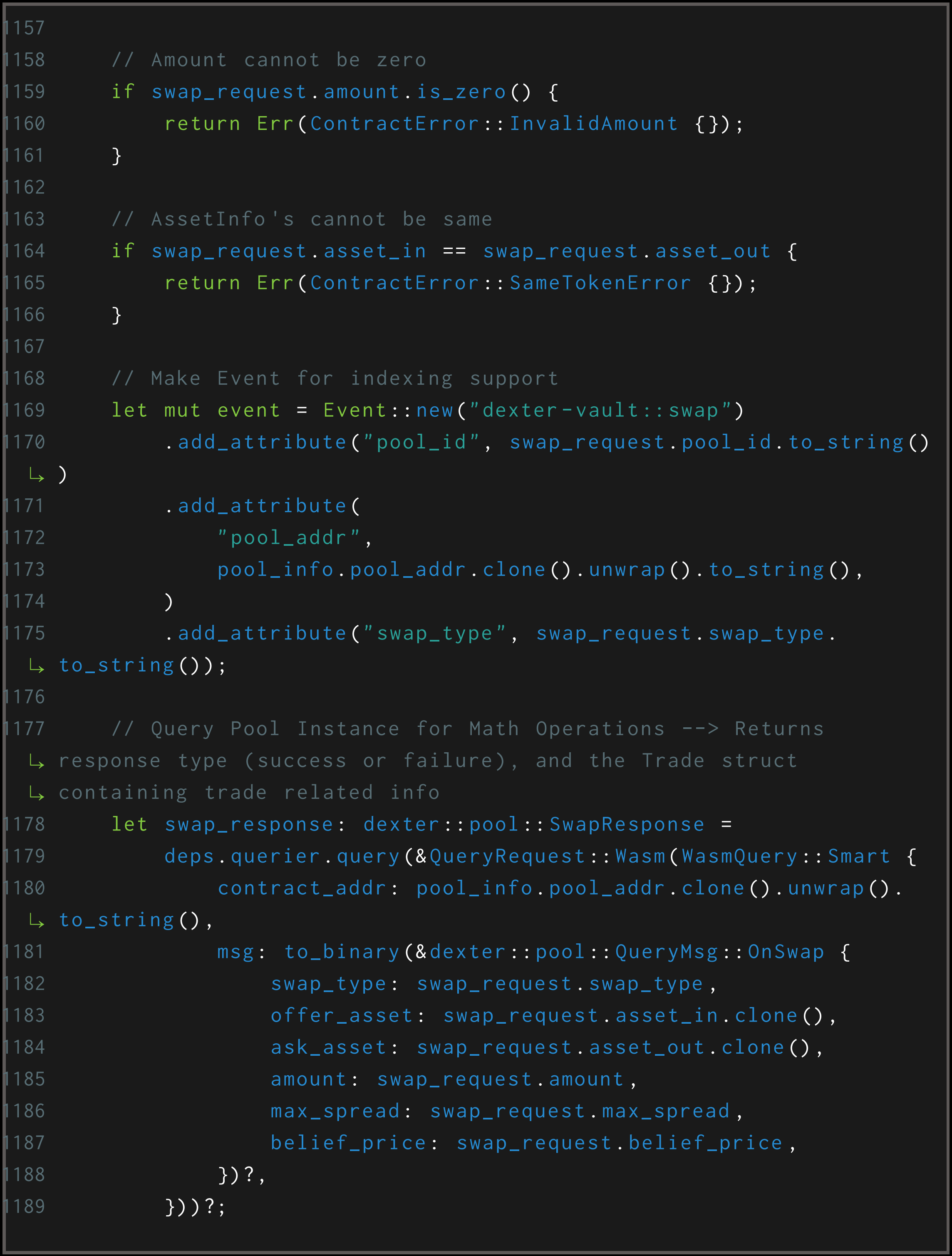
When using operation Swap from the vault contract, users can specify the value of min_receive which is the minimal amount of tokens that the user will receive from the swap. This protects the user from slippage, as if the tokens to be received as less than the user accepts as minimum, the swap will not succeed. However, the parameter is optional, so the user may not specify it, which may result in not being protected from potential slippage. Thus, it is important to ensure a default value of maximal accepted slippage, so even if the user does not choose a value, the potential slippage will be limited.

Code Location:

The below code shows a call to Swap operation containing optional value of minimal amount to receive, as well as other swap data in SingleSwapRequest struct:

Listing 2: contracts/vault/src/contract.rs (Lines 1146,1148)

```
1142     pub fn execute_swap(
1143     deps: DepsMut,
1144     env: Env,
1145     info: MessageInfo,
1146     swap_request: SingleSwapRequest,
1147     op_recipient: Option<String>,
1148     min_receive: Option<Uint128>,
1149     max_spend: Option<Uint128>,
1150 ) -> Result<Response, ContractError> {
1151     // Load Pool Info from Storage
1152     let mut pool_info = ACTIVE_POOLS
1153         .load(deps.storage, swap_request.pool_id.to_string().
   ↳ as_bytes())
1154         .expect("Invalid Pool Id");
1155
1156     let config = CONFIG.load(deps.storage)?;
```



```
    // Amount cannot be zero
    if swap_request.amount.is_zero() {
        return Err(ContractError::InvalidAmount {});
    }

    // AssetInfo's cannot be same
    if swap_request.asset_in == swap_request.asset_out {
        return Err(ContractError::SameTokenError {});
    }

    // Make Event for indexing support
    let mut event = Event::new("dexter-vault::swap")
        .add_attribute("pool_id", swap_request.pool_id.to_string())
        .add_attribute(
            "pool_addr",
            pool_info.pool_addr.clone().unwrap().to_string(),
        )
        .add_attribute("swap_type", swap_request.swap_type.to_string());

    // Query Pool Instance for Math Operations --> Returns response type (success or failure), and the Trade struct containing trade related info
    let swap_response: dexter::pool::SwapResponse =
        deps.querier.query(&QueryRequest::Wasm(WasmQuery::Smart {
            contract_addr: pool_info.pool_addr.clone().unwrap().to_string(),
            msg: to_binary(&dexter::pool::QueryMsg::OnSwap {
                swap_type: swap_request.swap_type,
                offer_asset: swap_request.asset_in.clone(),
                ask_asset: swap_request.asset_out.clone(),
                amount: swap_request.amount,
                max_spread: swap_request.max_spread,
                belief_price: swap_request.belief_price,
            })?,
        }))?;
```

The min_receive parameter triggers a token amount check on swap. If it is missing, which is possible since the parameter is optional, the check is not triggered.

Listing 3: packages/dexter/src/vault.rs

```
    // If the `min_receive` amount is provided, then check if the ask asset amount is greater than the min receive amount and if not, then return error
    if min_receive.is_some() &&  min_receive.unwrap() > ask_asset.amount.clone() {
        return Err(ContractError::MinReceiveError { min_receive: min_receive.unwrap(), ask_amount: ask_asset.amount.clone()});
    }
}
```

Further, the SingleSwapRequest struct contains information on max_spread and belief_price which are also optional:

Listing 4: packages/dexter/src/vault.rs (Lines 202,203)

```
196     pub struct SingleSwapRequest {
197     pub pool_id: Uint128,
198     pub asset_in: AssetInfo,
199     pub asset_out: AssetInfo,
200     pub swap_type: SwapType,
201     pub amount: Uint128,
202     pub max_spread: Option<Decimal>,
203     pub belief_price: Option<Decimal>,
204 }
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

It is recommended to protect users against slippage if they do not choose any parameters that limit it. Ideally, parameters that influence receiving token amount such as belief_price, max_spread or min_receive could have a default value when not specified. These values can also be injected into the front-end, since that's where users will interact with the protocol. When users add liquidity or swap, but do not specify slippage tolerance (or its equivalent) or slippage value is greater than the threshold, a predefined, default value should be used. As a reference, **max slippage** for Uniswap Pool and Uniswap Swap is 50%.

Remediation Plan:

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

3.3 (HAL-03) FUNDS WILL BE LOST IF ADDITIONAL NATIVE TOKENS ARE SENT - MEDIUM

Description:

One of the functionalities of router multihop swap utility allows the first token of the swap to be native. In such case, the token should be attached to the ExecuteMsg as part of funds vector. If the amount of coins sent exceeds amount of tokens to be swapped defined in parameter offer_amount, then the surplus amount is returned to the sender. However, the function does not check if there are other native tokens attached to the request. Since CosmWasm allows multiple native tokens to be attached to one message, if any other tokens are sent in the same request, for example mistakenly of a user, they will not be returned and will be stuck in the contract forever, especially because there is no any withdraw utility implemented in the contract.

Code Location:

Below code shows the part of the code responsible for returning surplus amount of tokens to the sender. First, the incoming multiswap_request is passed to function get_sent_native_token_balance:

Listing 5: contracts/router/src/contract.rs

```
if multiswap_request[0].asset_in.is_native_token() {
    // Query - Get number of offer asset (Native) tokens sent with the msg
    let tokens_received = multiswap_request[0]
        .asset_in
        .get_sent_native_token_balance(&info);
```

The function uses Iterator::find to find the balance of sent asset. Note, that if there are more than one coins attached, the surplus ones will be ignored and just received.

Listing 6: packages/dexter/src/asset.rs

```

/// Returns the number of native tokens being sent
pub fn get_sent_native_token_balance(&self, message_info: & MessageInfo) -> Uint128 {
    if let AssetInfo::NativeToken { denom } = &self {
        match message_info.funds.iter().find(|x| x.denom == *denom) {
            Some(coin) => {
                return coin.amount;
            }
            None => {
                return Uint128::zero();
            }
        }
    } else {
        return Uint128::zero();
    }
}
```

In the last step, the below piece of code calculates amount of surplus tokens. It considers only the denom that matched the asset_in, anything others will not be sent back.

Listing 7: contracts/router/src/contract.rs (Line 168)

```
164     // ExecuteMsg -if the number of native tokens sent is greater than the offer amount, then send the remaining tokens back to the sender
165     if tokens_received > offer_amount {
166         execute_msgs.push(multiswap_request[0].asset_in.clone().into_msg(
167             info.sender.clone(),
168             tokens_received.checked_sub(offer_amount)?,
169         )?);
170     }
171 }
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

To solve that issue, it might be worth to reject Swap requests that have multiple native tokens attached.

Remediation Plan:

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

3.4 (HAL-04) INADEQUATE ACCESS CONTROL PREVENTS UPDATING THE POOL FEE - MEDIUM

Description:

The execute_update_pool_config function from **vault** contract allows updating the fee_info.total_fee_bps parameter in any of the current active pools. This function first updates the PoolInfo state stored in the vault, so to update the information in the pool itself, the function executes an ExecuteMsg::UpdateFee transaction from the vault to the target pool.

In the target pool, the function to be executed is update_total_fee_bps, whose access control limits the execution to the vault owner only. Since this execution comes from the vault itself, it will be rejected and the fee parameter will not be updated. In order to update the parameter, the vault owner would have to perform the transaction himself.

This issue applies only to Stable and Stable5 pools, however, the XYK and weighted pools have correct access control that allows both: the vault address and the vault owner.

Code Location:

Fragment of execute_update_pool_config from **vault** contract:

Listing 8: contracts/vault/src/contract.rs (Line 501)

```
497 // update total fee in the actual pool contract by sending a wasm
  ↳ message
498 let msg = CosmosMsg::Wasm(WasmMsg::Execute {
499     contract_addr: pool.pool_addr.to_string(),
500     funds: vec![],
501     msg: to_binary(&dexter::pool::ExecuteMsg::UpdateFee {
502         total_fee_bps: pool.fee_info.total_fee_bps.clone(),
503     })?,
504 });
```

Fragment of the update_total_fee_bps function from **Stable5** and **Stable** pool contracts:

Listing 9: contracts/pools/stable_5pool/src/contract.rs (Lines 256-258)

```
246 pub fn update_total_fee_bps(
247     deps: DepsMut,
248     _env: Env,
249     info: MessageInfo,
250     total_fee_bps: u16,
251 ) -> Result<Response, ContractError> {
252     let mut config = CONFIG.load(deps.storage)?;
253     let vault_config = query_vault_config(&deps.querier, config.vault_addr.clone().to_string())?;
254
255     // Access Check :: Only Vault's Owner can execute this function
256     if info.sender != vault_config.owner {
257         return Err(ContractError::Unauthorized {});
258     }
259
260     config.fee_info.total_fee_bps = total_fee_bps;
261     CONFIG.save(deps.storage, &config)?;
262
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

It is recommended to add the vault address in the access control of the update_total_fee_bps function in Stable and Stable5 pools.

Remediation Plan:

SOLVED: The Dexter team has solved this issue in commit 830cbce.

3.5 (HAL-05) SOME PARAMETERS MAY BE OVERWRITTEN WHEN UPDATING THE VAULT CONFIGURATION - MEDIUM

Description:

When the owner of the vault contract wants to update any setting from the vault configuration, the owner sends an ExecuteMsg::UpdateConfig message. This message is composed of optional fields, which are checked later in the update_config function, in order to see which settings will be updated.

However, there is not a check to see if the auto_stake_impl field has been sent or not. This setting is always updated when the contract process an **UpdateConfig** message. Because of this situation, if the message does not contain the auto_stake_impl field, the value of auto_stake_imp is **None**, which turns off the auto-stake implementation.

This also happens with the pool creation fee. There is a check to see if the field pool_creation_fee has been sent in the message, but the value in the config is updated outside this check, so even if the message does not contain the pool_creation_fee, the value is updated anyway, setting its value to **None**.

As a consequence, the owner could unintentionally shut down the autostake implementation and/or remove the pool creation fee.

Code Location:

Fragment of update_config function from **vault** contract:

Listing 10: contracts/vault/src/contract.rs (Lines 353,356,361)

```
352 // set auto stake implementation
353 config.auto_stake_impl = auto_stake_impl;
354
355 // Validate the pool creation fee
```

```
if let Some(pool_creation_fee) = &pool_creation_fee {
    if pool_creation_fee.amount.is_zero() {
        return Err(ContractError::InvalidPoolCreationFee {});
    }
}
config.pool_creation_fee = pool_creation_fee;

```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

It is recommended to add some checks in place to update the auto_stake_impl and the pool_creation_fee **only** when those fields are sent in the **ExecuteMsg::UpdateConfig message**.

Remediation Plan:

SOLVED: The Dexter team has solved this issue in commit 6f3216c.

3.6 (HAL-06) GIVEOUT SWAP TYPE CANNOT BE EXECUTED - LOW

Description:

The multi-hop swap router allows for simulating two types of swaps: GiveIn and GiveOut. They differ in logic, as GiveIn performs swaps based on amount of asset provided to the swap, and GiveOut bases on amount expected to be returned from the swap. Originally, the core swap-performing component, vault, allows for both types of swaps. When performing a single swap via vault, and not a multi-hop swap, user can choose GiveOut or GiveIn type swap. Users can also simulate both types of swaps using router’s function query_simulate_multihop. However, the swap functionality, when accesses via MultiHopRouter allows only GiveIn swap types due to it is hardcoded it the code. Users who wish to perform GiveOut type multi-swap can simulate it, but not execute it.

Code Location:

Below code shows the entry point for multi-hop swap in function execute_multihop_swap. It passes user arguments to SingleSwapRequest struct, which is a part of first_hop_swap_request variable now:

Listing 11: contracts/router/src/contract.rs (Line 202)

```
196 // Create SingleSwapRequest for the first hop
197 let first_hop = multiswap_request[0].clone();
198 let first_hop_swap_request = SingleSwapRequest {
199     pool_id: first_hop.pool_id,
200     asset_in: first_hop.asset_in.clone(),
201     asset_out: first_hop.asset_out.clone(),
202     swap_type: SwapType::GiveIn {},
203     // Amount provided is the amount to be used for the first hop
204     amount: offer_amount,
205     max_spread: first_hop.max_spread,
206     belief_price: first_hop.belief_price,
207 };
```

Next, the first_hop_swap_request is used to perform a Swap, and the swap_type is taken from above hardcoded value:

Listing 12: contracts/router/src/contract.rs (Line 228)

```
223 // ExecuteMsg - For the first hop
224 let first_hop_execute_msg = CosmosMsg::Wasm(WasmMsg::Execute {
225     contract_addr: config.dexter_vault.to_string(),
226     funds: coins,
227     msg: to_binary(&vault::ExecuteMsg::Swap {
228         swap_request: first_hop_swap_request.clone(),
229         recipient: Some(env.contract.address.clone().to_string()),
230         min_receive: None,
231         max_spend: None,
232     })?,
233 });
234 execute_msgs.push(first_hop_execute_msg);
```

For example, the query_simulate_multihop recognizes two cases of simulated swap, one is GiveIn and one GiveOut swap type. The code for handling GiveOut type swap simulation of that function is reflected in the below code snippet:

Listing 13: contracts/router/src/contract.rs (Line 549)

```
544 // Query pool to get the swap transition response
545  let pool_swap_transition: dexter::pool::SwapResponse =
546      deps.querier.query(&QueryRequest::Wasm(WasmQuery::Smart {
547          contract_addr: pool_response.pool_addr.clone().unwrap().
  ↳ to_string(),
548          msg: to_binary(&dexter::pool::QueryMsg::OnSwap {
549              swap_type: SwapType::GiveOut {},
550              offer_asset: hop.asset_in.clone(),
551              ask_asset: hop.asset_out.clone(),
552              amount: prev_amount_out.clone(),
553              max_spread: hop.max_spread,
554              belief_price: hop.belief_price,
555          })?,
556      }))?;
```

Risk Level:

Likelihood - 4
Impact - 1

Recommendation:

It is recommended to either allow performing GiveOut type swaps via the router or disable GiveOut type query simulation because currently users can simulate that type of swap but not execute it.

Remediation plan:

NOT APPLICABLE: The Dexter team stated that the described behavior of the function is as intended and the reason for not allowing GiveOut multi-hop swaps is that it is not feasible as it would be too complex and gas heavy for a transaction. At the same time, it is desired to allow the simulation of those scenarios, since users may want to know an approximation of how many input tokens they need to spend to get a particular amount of output.

3.7 (HAL-07) INSUFFICIENT LP TOKEN VALIDATION - LOW

Description:

The create_pool capability allows for creating new pools, and it takes lp_token_name and lp_token_symbol as parameters. These parameters are undergoing a basic validation which checks their length, but this makes not be sufficient to ensure optimal security of the protocol. Two main attack vectors are possible: first, embedding scripts into the strings, as they are not checked against not beginning alphanumeric; and second - possible social engineering / scam attempt by creating a duplicate LP token names to mimic more valuable ones.

Code Location:

The below code shows validation routines for LP token names and symbols:

Listing 14: packages/dexter/src/helper.rs (Lines 461,473)

```rust
pub fn is_valid_name(name: &str) -> bool {
    let bytes = name.as_bytes();
    if bytes.len() < 3 || bytes.len() > 50 {
        return false;
    }
    true
}

pub fn is_valid_symbol(symbol: &str) -> bool {
    let bytes = symbol.as_bytes();
    if bytes.len() < 3 || bytes.len() > 12 {
        return false;
    }
    for byte in bytes.iter() {
        if (*byte != 45) && (*byte < 65 || *byte > 90) && (*byte < 97 || *byte > 122) {
            return false;
        }
    }
    true
}
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

It is recommended to validate the strings against a function like is_alphanumeric(). Moreover, the most secure path would be to generate LP token names and symbols automatically out of underlying assets instead of allowing users to specify arbitrary names.

Remediation Plan:

SOLVED: The Dexter team has solved this issue in commit 132eeee.

3.8 (HAL-08) INSUFFICIENT XML VALIDATION - LOW

Description:

The lp_token contract allows for specifying marketing info, in form of an XML or PNG logo. The XML one may be vulnerable to various Web2 attacks related to XML, among others XXE, XEE or more sophisticated attacks, which may lead to compromise of the underlying dApp. Allowing XML supplied by users should always be performed with extreme caution. The code in the contract does not perform any sanity check on the XML code apart from a size check, potentially allowing malicious DTD declarations to be supplied along the XML code.

Code Location:

Below code shows the verify_xml_preamble function, which checks if the document starts with a proper xml declaration. However, checking for DTD elements is not present.

Listing 15: contracts/lp_token/src/contract.rs

```rust
/// Checks if data starts with XML preamble
fn verify_xml_preamble(data: &[u8]) -> Result<(), ContractError> {
    // The easiest way to perform this check would be just match on regex, however regex  compilation is heavy and probably not worth it.
    let preamble = data
        .split_inclusive(|c| *c == b'>')
        .next()
        .ok_or(ContractError::InvalidXmlPreamble {})?;

    const PREFIX: &[u8] = b"<?xml ";
    const POSTFIX: &[u8] = b"?>";

    if !(preamble.starts_with(PREFIX) && preamble.ends_with(POSTFIX)) {
        Err(ContractError::InvalidXmlPreamble {})
    } else {
        Ok(())
    }

    // Additionally attributes format could be validated as they are well-defined, as well as
    // comments presence inside preable, but it is probably not worth it.
}

/// Validates XML logo
fn verify_xml_logo(logo: &[u8]) -> Result<(), ContractError> {
    verify_xml_preamble(logo)?;

    if logo.len() > LOGO_SIZE_CAP {
        Err(ContractError::LogoTooBig {})
    } else {
        Ok(())
    }
}
```

Risk Level:

Likelihood - 1
Impact - 3

Remediation Plan:

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

3.9 (HAL-09) LACK OF TRANSFER OWNERSHIP MECHANISM IN KEEPER CONTRACT - LOW

Description:

The **keeper** contract does not implement any functionality allowing to change the ownership. If in the future it turns out to be necessary, the change might require increased time and effort to process, especially due to the fact the contract is part of a larger dApp.

Code Location:

Below code of keeper contract shows the update_config function, which does not implement the ability to pass the ownership to another owner.

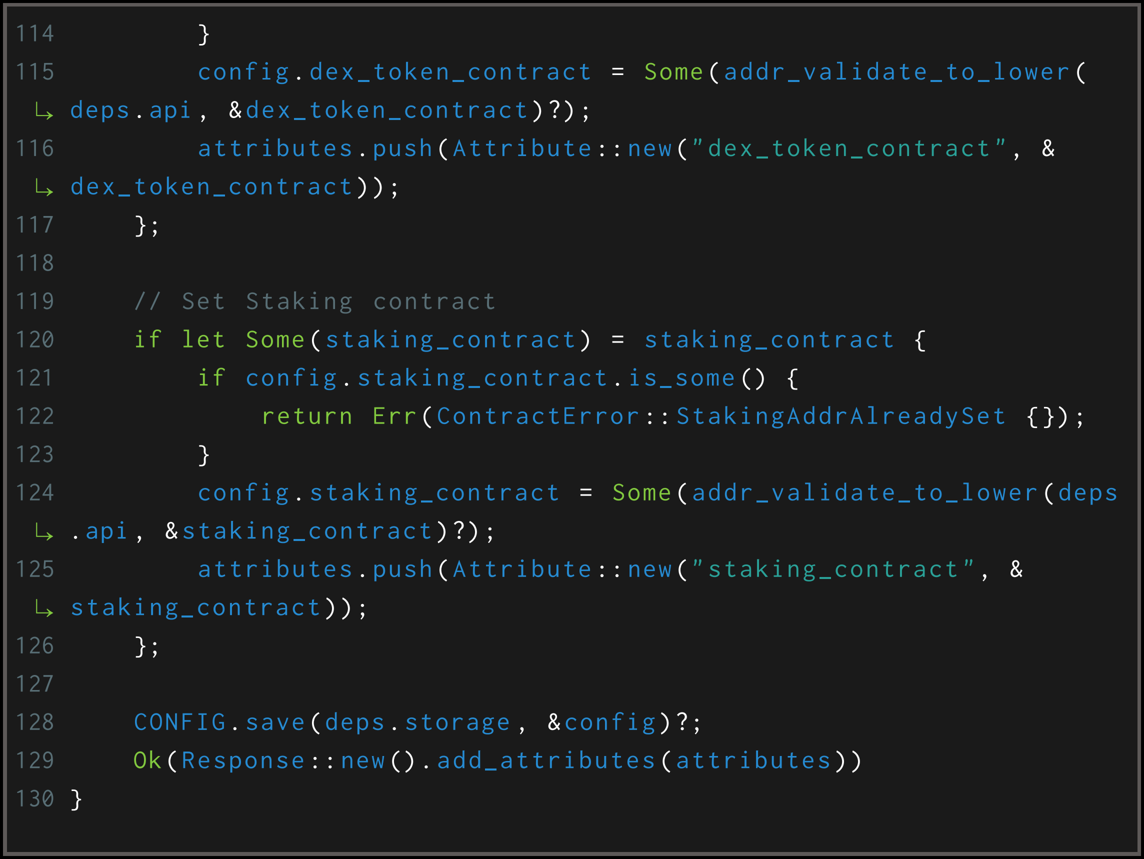
Listing 16: contracts/keeper/src/contract.rs

```rust
fn update_config(
    deps: DepsMut,
    info: MessageInfo,
    dex_token_contract: Option<String>,
    staking_contract: Option<String>,
) -> Result<Response, ContractError> {
    let mut attributes = vec![attr("action", "set_config")];

    let mut config = CONFIG.load(deps.storage)?;
    let vault_config_res = query_vault_config(&deps.querier,
 ↳ config.vault_contract.to_string())?;

    // Permission check
    if info.sender != vault_config_res.owner {
        return Err(ContractError::Unauthorized {});
    }

    // Set DEX token contract
    if let Some(dex_token_contract) = dex_token_contract {
        if config.dex_token_contract.is_some() {
            return Err(ContractError::DexTokenAlreadySet {});
```



```
        }
        config.dex_token_contract = Some(addr_validate_to_lower(deps.api, &dex_token_contract)?);
        attributes.push(Attribute::new("dex_token_contract", &dex_token_contract));
    };

    // Set Staking contract
    if let Some(staking_contract) = staking_contract {
        if config.staking_contract.is_some() {
            return Err(ContractError::StakingAddrAlreadySet {});
        }
        config.staking_contract = Some(addr_validate_to_lower(deps.api, &staking_contract)?);
        attributes.push(Attribute::new("staking_contract", &staking_contract));
    };

    CONFIG.save(deps.storage, &config)?;
    Ok(Response::new().add_attributes(attributes))
}
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is recommended to implement a safe transfer ownership logic.

Remediation Plan:

SOLVED: The Dexter team has solved this issue in commit ff80422 by adding an ownership transfer procedure.

3.10 (HAL-10) EMERGENCY UNSTAKE DO NOT CHECKS FOR EMERGENCY STATUS - LOW

Description:

The emergency_unstake function of the **generator** contract allows withdrawing the stake deposited in the contract if some kind of emergency occurs, however, no state has been declared for emergencies, so this function could be called at any time by any user.

Code Location:

Fragment of emergency_unstake function from **generator** contract:

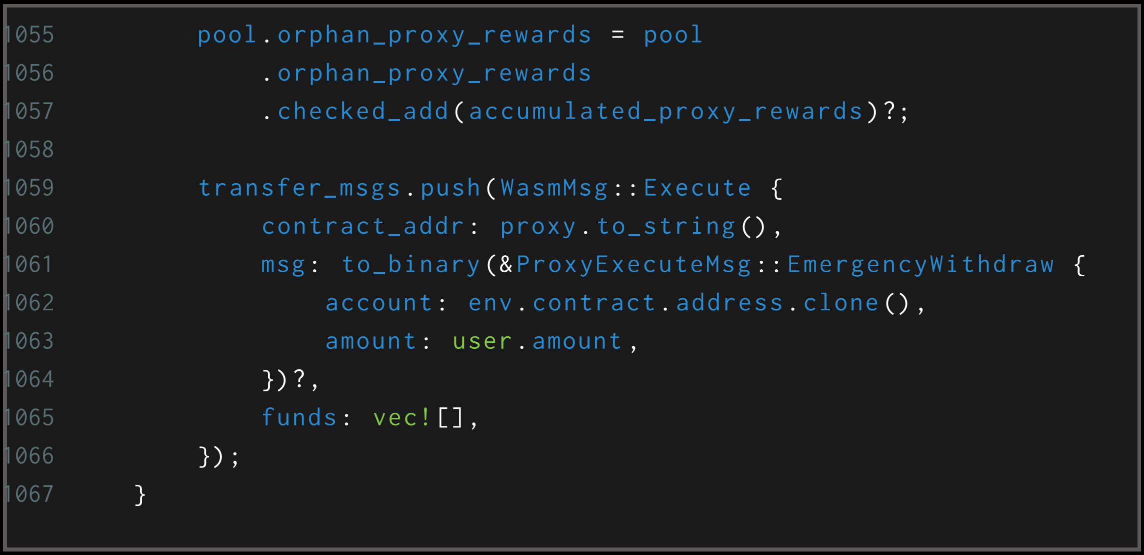
Listing 17: contracts/dexter_generator/generator/src/contract.rs

```rust
pub fn emergency_unstake(
    deps: DepsMut,
    env: Env,
    info: MessageInfo,
    lp_token: String,
) -> Result<Response, ContractError> {
    let lp_token = deps.api.addr_validate(&lp_token)?;
    let cfg = CONFIG.load(deps.storage)?;

    let mut pool = POOL_INFO.load(deps.storage, &lp_token)?;
    let user = USER_INFO.load(deps.storage, (&lp_token, &info.sender.clone()))?;

    // Instantiate the transfer call for the LP token
    let mut transfer_msgs: Vec<WasmMsg> = vec![];
    if let Some(proxy) = &pool.reward_proxy {
        let accumulated_proxy_rewards = pool
            .accumulated_proxy_rewards_per_share
            .checked_mul_uint128(user.amount)?
            .checked_sub(user.reward_debt_proxy)?;

        // All users' proxy rewards become orphaned
```



```
        pool.orphan_proxy_rewards = pool
            .orphan_proxy_rewards
            .checked_add(accumulated_proxy_rewards)?;

        transfer_msgs.push(WasmMsg::Execute {
            contract_addr: proxy.to_string(),
            msg: to_binary(&ProxyExecuteMsg::EmergencyWithdraw {
                account: env.contract.address.clone(),
                amount: user.amount,
            })?,
            funds: vec![],
        });
    }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is recommended to add a new state variable to declare an owner-managed emergency state in order to avoid any unintentional execution of this function.

Remediation Plan:

SOLVED: The Dexter team has solved this issue, since the **generator** contract has been removed from the final version of the code.

3.11 (HAL-11) IMPROPER ERROR HANDLING (MULTISTAKING) - LOW

Description:

The unlock function of the **multi_staking** contract allows withdrawing the previously unlocked participation after the unlock_period has elapsed. The variable USER_LP_TOKEN_LOCKS stores for each user the number of LP tokens to unlock at a given time.

Since the lp_token input comes directly from the execute call, in case of any error, the function that should load the status will create an empty one with invalid data. This is due to the use of unwrap_or_default() instead of unwrap_or_else(ERROR_MSG).

The function ends up returning an empty response, while the USER_LP_TOKEN_LOCKS state stores a useless state with a wrong lp_token address.

Code Location:

Fragment of unlock function from **multi_staking** contract:

Listing 18: contracts/multi_staking/src/contract.rs (Lines 525,538)

```
522 pub fn unlock(deps: DepsMut, env: Env, sender: Addr, lp_token:
    ↳ Addr) -> ContractResult<Response> {
523     let locks = USER_LP_TOKEN_LOCKS
524         .may_load(deps.storage, (&lp_token, &sender))?
525         .unwrap_or_default();
526
527     let mut response = Response::new();
528     let total_unlocked_amount = locks
529         .iter()
530         .filter(|lock| lock.unlock_time <= env.block.time.seconds
    ↳ ())
531         .fold(Uint128::zero(), |acc, lock| acc + lock.amount);
532
533     let updated_unlocks = locks
```

```
        .into_iter()
        .filter(|lock| lock.unlock_time > env.block.time.seconds())
        .collect::<Vec<TokenLock>>();

    USER_LP_TOKEN_LOCKS.save(deps.storage, (&lp_token, &sender), &updated_unlocks)?;

    if total_unlocked_amount.is_zero() {
        return Ok(response);
    }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is recommended to replace unwrap_or_default() with unwrap_or_else(ERROR_MSG) to avoid storing erroneous information into the contract.

Remediation Plan:

SOLVED: The Dexter team has solved this issue in commit 830cbce by avoiding saving the state created by default.

3.12 (HAL-12) EMPTY MIGRATE FUNCTION - INFORMATIONAL

Description:

The migrate() function code takes any MigrateMsg (with any code) returning OK() in all circumstances. There is no event logging (in the form of response attributes) which decreases visibility of a migration that happened. It should also be noted that any owner account that is the deployer of such an upgradable contract should be additionally protected, in the form of a multisignature wallet or governance, since upgrading a contract allows for entirely changing its codebase and in compromise scenario case allows performing any malicious action, including draining funds and interacting with other contracts.

Code Location:

The empty migrate function is present in several contracts:

- dexter_generator/generator/src/contract.rs
- dexter_generator/generator_proxy/src/contract.rs
- dexter_generator/ref_staking/src/contract.rs
- dexter_generator/vesting/src/contract.rs
- /lp_token/src/contract.rs

The below code shows an example migration that is handled in the same way in each other contract specified below. The migration has just a wildcard function, the contract version is not checked, and there is no information on the migration logged as response attributes.

Listing 19: contracts/dexter_generator/generator_proxy/src/contract.rs

```rust
pub fn migrate(_deps: DepsMut, _env: Env, _msg: MigrateMsg) ->
 ↳ StdResult<Response> {
    Ok(Response::default())
}
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

Implement attribute logging to increase visibility of migrations. To prevent potential migration abuses, contracts should be deployed using a DAO controlled address or a multisignature wallet, so that in the case of a single deployer compromise, the code is not replaced. Good practices for contract migration are described in the CosmWasm documentation - https://docs.cosmwasm.com/docs/1.0/smart-contracts/migration/.

Remediation Plan:

ACKNOWLEDGED: The Dexter team acknowledged this finding.

3.13 (HAL-13) LACK OF INPUT VALIDATION (MULTISTAKING) - INFORMATIONAL

Description:

The bond and unbond functions of the **multi_staking** contract do not perform some basic checks on the amount input in order to avoid useless operations or an underflow error.

It is not checked whether this amount is zero and, in the case of the unbond function, it is not checked whether the input amount is greater than the value of current_bond_amount.

Code Location:

Fragment of bond function form **multi_staking** contract:

Listing 20: contracts/multi_staking/src/contract.rs (Lines 364,371)

```
363 // Increase bond amount
364 lp_global_state.total_bond_amount = lp_global_state.
  ↳ total_bond_amount.checked_add(amount)?;
365 LP_GLOBAL_STATE.save(deps.storage, &lp_token, &lp_global_state)?;
366
367 // Increase user bond amount
368 USER_BONDED_LP_TOKENS.save(
369     deps.storage,
370     (&lp_token, &user),
371     &(current_bond_amount.checked_add(amount)?),
372 )?;
373
374 Ok(response)
```

Fragment of unbond function form **multi_staking** contract:

Listing 21: contracts/multi_staking/src/contract.rs (Lines 416,428)

```
413 USER_BONDED_LP_TOKENS.save(
414     deps.storage,
415     (&lp_token, &sender),
416     &(current_bond_amount.checked_sub(amount)?),
417 )?;
418
419 // Start unlocking clock for the user's LP Tokens
420 let mut unlocks = USER_LP_TOKEN_LOCKS
421     .may_load(deps.storage, (&lp_token, &sender))?
422     .unwrap_or_default();
423
424 let config = CONFIG.load(deps.storage)?;
425
426 unlocks.push(TokenLock {
427     unlock_time: env.block.time.seconds() + config.unlock_period,
428     amount,
429 });
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

It is recommended to add the proper validations to avoid the above-mentioned situations: amount should be greater than zero and, in case of unbound operation, it should be checked that current_bond_amount >= amount.

Remediation Plan:

SOLVED: The Dexter team solved this issue in commits c551886 and 5947d7a.

3.14 (HAL-14) WRONG DATA INPUT - INFORMATIONAL

Description:

The accumulate_prices function from **Weighted Pool** contract uses the time reference (block_time_last) from the config parameter instead of the twap variable as the rest of the pools.

This would affect the event raised in the execute_update_pool_liquidity function because the variable twap.block_time_last is never updated.

In addition, the _math_config input parameter is never used.

Code Location:

Fragment of accumulate_prices function from **Weighted Pool** contract:

Listing 22: contracts/pools/weighted_pool/src/utils.rs (Lines 161,164,195)

```
151 pub fn accumulate_prices(
152     deps: Deps,
153     env: Env,
154     config: &mut Config,
155     _math_config: MathConfig,
156     twap: &mut Twap,
157     pools: &[DecimalAsset],
158 ) -> Result<(), ContractError> {
159     // Calculate time elapsed since last price update.
160     let block_time = env.block.time.seconds();
161     if block_time <= config.block_time_last {
162         return Ok(());
163     }
164     let time_elapsed = Uint128::from(block_time - config.
  ↳ block_time_last);
```

End of the accumulate_prices function from **Weighted Pool** contract:

Listing 23: contracts/pools/weighted_pool/src/utils.rs (Line 195)

```
194     // Update last block time.
195     config.block_time_last = block_time;
196     Ok(())
197 }
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

It is recommended to use the twap variable for time reference instead of config one in accumulate_prices function of **Weighted Pool**.

Remediation Plan:

SOLVED: The Dexter team has solved this issue in commit d2db2ab replacing the config.block_time_last variable by twap.block_time_last one. They have also removed the config and _math_config parameters from the signature of the function because they are not used anymore.

3.15 (HAL-15) NON-ZERO CHECK PERFORMED AFTER TRANSACTION - INFORMATIONAL

Description:

In the emergency_unstake function of the **generator** contract, the non-zero check is performed after some operations take place.

This check should be done at the beginning of the function to handle the error correctly and save some gas.

Code Location:

Fragment of emergency_unstake function from **generator** contract:

Listing 24: contracts/dexter_generator/generator/src/contract.rs (Lines 1051,1063,1070,1074-1076)

```
1043     let mut pool = POOL_INFO.load(deps.storage, &lp_token)?;
1044     let user = USER_INFO.load(deps.storage, (&lp_token, &info.
  ↳ sender.clone()))?;
1045
1046     // Instantiate the transfer call for the LP token
1047     let mut transfer_msgs: Vec<WasmMsg> = vec![];
1048     if let Some(proxy) = &pool.reward_proxy {
1049         let accumulated_proxy_rewards = pool
1050             .accumulated_proxy_rewards_per_share
1051             .checked_mul_uint128(user.amount)?
1052             .checked_sub(user.reward_debt_proxy)?;
1053
1054         // All users' proxy rewards become orphaned
1055         pool.orphan_proxy_rewards = pool
1056             .orphan_proxy_rewards
1057             .checked_add(accumulated_proxy_rewards)?;
1058
1059         transfer_msgs.push(WasmMsg::Execute {
1060             contract_addr: proxy.to_string(),
1061             msg: to_binary(&ProxyExecuteMsg::EmergencyWithdraw {
```

```
                account: env.contract.address.clone(),
                amount: user.amount,
            })?,
            funds: vec![],
        });
    }

    // Update user's balance. All LP tokens are to be unbonded and the user's bonded amount is set to 0.
    let unbonded_amount = user.amount;
    let mut user = update_user_balance(user, &pool, Uint128::zero())?;

    // Check that amount is non-zero
    if unbonded_amount == Uint128::zero() {
        return Err(ContractError::ZeroUnbondAmount {});
    }
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

It is recommended to move the piece of code where the user.amount is checked at the beginning of the function.

Remediation Plan:

SOLVED: The Dexter team solved this issue in commit 830cbce.

3.16 (HAL-16) LACK OF ADDRESS VALIDATION - INFORMATIONAL

Description:

Address validation is a basis for good practice in smart contract development. A malformed address stored in the code would cause future errors during execution. This problem has an easy solution, which is to validate each address before storing or using it.

This problem has appeared in several contracts and functions listed below.

Contract	Function	Address
multi_staking	allow_lp_token	lp_token
keeper	instantiate	msg.owner
keeper	withdraw	recipient
vesting	claim	recipient
router	execute_multihop_swap	recipient
vault (new commit)	instantiate	auto_stake_impl
vault (new commit)	update_config	auto_stake_impl

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

It is recommended to validate each address before storing or using it.

Remediation Plan:

SOLVED: The Dexter team has solved this issue:

Contract	Function	Address	Solved?
multi_staking	allow_lp_token	lp_token	Yes (830cbce)
keeper	instantiate	msg.owner	Yes (0c199cd)
keeper	withdraw	recipient	Yes (830cbce)
vesting	claim	recipient	Yes (830cbce)
router	execute_multihop_-swap	recipient	Yes (830cbce)
vault (new commit)	instantiate	auto_stake_impl	Yes (0c199cd)
vault (new commit)	update_config	auto_stake_impl	Yes (0c199cd)

3.17 (HAL-17) WRONG IMPLEMENTATION CAN WASTE SOME GAS (HELPER) - INFORMATIONAL

Description:

The select_pools function from **package/dexter/src/helpers.rs** is used in some project contracts to find and select a specific pool depending on the offer/ask pair entry.

For pools with 2 assets, the function behaves differently from other pools, checking one of the assets and taking the other automatically using the modulus of 2 (% 2) function. This means that for pools with two assets it does not detect if the user has submitted an erroneous “ask” asset, as the checks stop on the offered one.

The error will appear later in the vault, after all calculations. Looking for this error at the beginning of the function would save some gas. Also, this function is never used with optional input parameters, so the signature of the function could be changed.

Code Location:

select_pools function from **package/dexter/src/helpers.rs** file:

Listing 25: packages/dexter/src/helper.rs (Lines 212-217)

```
204 pub fn select_pools(
205     offer_asset_info: Option<&AssetInfo>,
206     ask_asset_info: Option<&AssetInfo>,
207     pools: &[DecimalAsset],
208 ) -> Result<(DecimalAsset, DecimalAsset), ContractError> {
209     // if pool is only contains 2 assets
210     if pools.len() == 2 {
211         match (offer_asset_info, ask_asset_info) {
212             (Some(offer_asset_info), _) => {
213                 let (offer_ind, offer_pool) = pools
214                     .iter()
```

```
          .find_position(|pool| pool.info.eq(offer_asset_info))
          .ok_or(ContractError::AssetMismatch {})?;
        Ok((offer_pool.clone(), pools[(offer_ind + 1) % 2].clone()))
      }
      (_, Some(ask_asset_info)) => {
        let (ask_ind, ask_pool) = pools
          .iter()
          .find_position(|pool| pool.info.eq(ask_asset_info))
          .ok_or(ContractError::AssetMismatch {})?;
        Ok((pools[(ask_ind + 1) % 2].clone(), ask_pool.clone()))
      }
      _ => Err(ContractError::VariableAssetMissed {}), // Should always be unreachable
    }
  } else if let (Some(offer_asset_info), Some(ask_asset_info)) =
    (offer_asset_info, ask_asset_info)
  {
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

It is recommended to check if both assets belong to the pool before finalizing the function.

Remediation Plan:

SOLVED: The Dexter team has solved this issue in commit 830cbce.

3.18 (HAL-18) EARLIER VALIDATION CAN SAVE SOME GAS (VAULT) - INFORMATIONAL

Description:

The execute_exit_pool function of the **vault** contract allows retrieving some ask-assets in exchange for burning some LP tokens (shares). Depending on the amount of ask-tokens requested, the amount of shares needed to burn will vary. This amount is known by a query made to the corresponding pool (query_on_exit_pool). If the burn_shares needed is zero, the function should return an error.

This is a simple check that could be done right after the query to the pool, however, in this case it is done almost 200 lines of code after the query. Moving the check to the beginning of the function would save some gas.

Code Location:

Pool query performed at the beginning of execute_exit_pool function:

Listing 26: contracts/vault/src/contract.rs (Line 1297)

```
let pool_exit_transition: dexter::pool::AfterExitResponse =
    deps.querier.query(&QueryRequest::Wasm(WasmQuery::Smart {
        contract_addr: pool_info.pool_addr.to_string(),
        msg: to_binary(&dexter::pool::QueryMsg::OnExitPool {
            assets_out,
            burn_amount,
        })?,
    }))?;
```

Burn_shares check performed at the end of execute_exit_pool function:

Listing 27: contracts/vault/src/contract.rs (Line 1484)

```
1483 // Check - Burn amount cannot be 0
1484 if pool_exit_transition.burn_shares.is_zero() {
1485     return Err(ContractError::BurnAmountZero {});
1486 }
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

It is recommended to move the non-zero burn_shares check to the beginning of the function to save some gas and gain efficiency.

Remediation Plan:

SOLVED: The Dexter team has solved this issue in commit 830cbce.

3.19 (HAL-19) INCOMPLETE MIGRATION - INFORMATIONAL

Description:

The keeper contract code contains preparation for a migration utility in the form of declaring a contract version and name, as well as corresponding comments. However, the migration is not implemented in it. If the contract is deployed and used in the current state, and a need for migration happens, it will not be possible. Replacing the migration with redeployment might require intense effort, and in the worst case, some desired actions may not be possible, e.g. moving funds out of the contract.

Code Location:

Below code shows the migration declaration, which is not followed by proper migration function.

Listing 28: contracts/keeper/src/contract.rs

```
/// Contract name that is used for migration.
const CONTRACT_NAME: &str = "dexter-keeper";
/// Contract version that is used for migration.
const CONTRACT_VERSION: &str = env!("CARGO_PKG_VERSION");
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to implement the migration functionality or remove the migration stub completely in case it is confirmed to be unintended in this contract.

Remediation Plan:

SOLVED: The issue was solved in commit f6f6548.

3.20 (HAL-20) OVERFLOW-CHECKS BEST PRACTICES - INFORMATIONAL

Description:

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

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

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to include the overflow-checks = true in the Cargo.toml of all the contracts.

Remediation Plan:

ACKNOWLEDGED: The Dexter team acknowledged this finding.

3.21 (HAL-21) USELESS CODE - INFORMATIONAL

Description:

The query_on_join_pool function of the **stable_pool** contract executes useless code in a couple of situations:

- The first one is the call to assert_slippage_tolerance function, which is a function that only returns the value Success without performing any operation.
- The second is the d_before_addition_liquidity >= d_after_addition_liquidity validation which, instead of returning an error message, calls to the zero function without any value being assigned to it.

Code Location:

Fragment of query_on_join_pool function from **stable_pool** contract:

Listing 29: contracts/pools/stable_pool/src/contract.rs (Line 420)

```
415 // Adjust deposit amounts to the precision of the pool tokens
416 let deposit_amount_0 = adjust_precision(deposits[0],
 ↳ token_precision_0, greater_precision)?;
417 let deposit_amount_1 = adjust_precision(deposits[1],
 ↳ token_precision_1, greater_precision)?;
418
419 // Assert slippage tolerance
420 let res = assert_slippage_tolerance(
421     &slippage_tolerance,
422     &deposits,
423     &[config.assets[0].amount, config.assets[1].amount],
424 );
425 // return a `Failure` response if the slippage tolerance is not
 ↳ met
426 if !res.is_success() {
427     return Ok(return_join_failure(res.to_string()));
428 }
```

assert_slippage_tolerance function from **stable_pool** contract:

Listing 30: contracts/pools/stable_pool/src/contract.rs (Line 1030)

```
1024 fn assert_slippage_tolerance(
1025     _slippage_tolerance: &Option<Decimal>,
1026     _deposits: &[Uint128; 2],
1027     _pools: &[Uint128; 2],
1028 ) -> ResponseType {
1029     // There is no slippage in the stable pool
1030     ResponseType::Success {}
1031 }
1032
```

Fragment of query_on_join_pool function from **stable_ťpool** contract:

Listing 31: contracts/pools/stable_pool/src/contract.rs (Lines 483-485)

```
479 let d_after_addition_liquidity =
480     compute_d(leverage, pool_amount_0.u128(), pool_amount_1.u128()
  ↳ ).unwrap();
481
482 // d after adding liquidity must be more than d before adding
  ↳ liquidity
483 if d_before_addition_liquidity >= d_after_addition_liquidity {
484     Uint128::zero();
485 }
486
487 total_share.multiply_ratio(
488     d_after_addition_liquidity - d_before_addition_liquidity,
489     d_before_addition_liquidity,
490 )
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to eliminate those pieces of code that do not contribute to the functionality of the system.

Remediation Plan:

SOLVED: The Dexter team has solved the issue of the d_before_addition_liquidity validation in commit 830cbce by removing the **stable_pool** contract from the final version of the code.

3.22 (HAL-22) UNNECESSARY PARAMETERS - INFORMATIONAL

Description:

The instantiate function of the **keeper** contract initializes the configuration parameters with the initial values. One of those values, the vault_address, is required in the initialization but never used.

The other two addresses configured during the instantiation (dex_token_contract and staking_contract) are only used in the update_config function and nowhere else.

Code Location:

Instantiate function from **keeper** contract:

Listing 32: contracts/keeper/src/contract.rs (Line 46)

```
36 pub fn instantiate(
37     deps: DepsMut,
38     _env: Env,
39     _info: MessageInfo,
40     msg: InstantiateMsg,
41 ) -> Result<Response, ContractError> {
42     set_contract_version(deps.storage, CONTRACT_NAME,
   ↳ CONTRACT_VERSION)?;
43
44     let cfg = Config {
45         owner: msg.owner,
46         vault_contract: deps.api.addr_validate(&msg.vault_contract
   ↳ )?,
47         dex_token_contract: None,
48         staking_contract: None,
49     };
50
51     CONFIG.save(deps.storage, &cfg)?;
52     Ok(Response::default())
53 }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to remove the unused parameters.

Remediation Plan:

SOLVED: The Dexter team has solved this issue in commit 7d11fad by removing all the unnecessary parameters of the **keeper** contract.

3.23 (HAL-23) USE OF MAGIC NUMBERS - INFORMATIONAL

Description:

The instantiate function of **stable_5pool** and **weighted_pool** contracts use magic numbers to set the limit on the number of assets a pool can have.

It is recommended to use constant variables instead.

Code Location:

instantiate function from **stable_5pool** contract:

Listing 33: contracts/pools/stable_5pool/src/contract.rs (Line 56)

```
47 pub fn instantiate(
48     mut deps: DepsMut,
49     env: Env,
50     _info: MessageInfo,
51     msg: InstantiateMsg,
52 ) -> Result<Response, ContractError> {
53     set_contract_version(deps.storage, CONTRACT_NAME,
 ↳ CONTRACT_VERSION)?;
54
55     // Validate number of assets
56     if msg.asset_infos.len() > 5 || msg.asset_infos.len() < 2 {
57         return Err(ContractError::InvalidNumberOfAssets {});
58     }
```

instantiate function from **weighted_pool** contract:

Listing 34: contracts/pools/weighted_pool/src/contract.rs (Line 63)

```
54 pub fn instantiate(
55     mut deps: DepsMut,
56     env: Env,
57     _info: MessageInfo,
58     msg: InstantiateMsg,
) -> Result<Response, ContractError> {
    set_contract_version(deps.storage, CONTRACT_NAME, CONTRACT_VERSION)?;

    // Validate number of assets
    if msg.asset_infos.len() > 8 || msg.asset_infos.len() < 2 {
        return Err(ContractError::InvalidNumberOfAssets {});
    }

```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to avoid the use of magic numbers to improve the readability of the code and to facilitate its maintenance, using constant values instead.

Remediation Plan:

SOLVED: The Dexter team has solved this issue for the **weighted_pool** contract in commit d2db2ab and for the **stable_5pool** in the commit 6773bfb by adding the MAX_ASSETS and MIN_ASSETS constants.

3.24 (HAL-24) UNCHECKED MATH - INFORMATIONAL

Description:

In the **ref_staking** contract, there are some functions that use unchecked mathematics in sum operations. Since these sum operations could work with large quantities, it is better to have a verified operation in case of future overflow.

This is a good practice, since having overflow-checks enabled in the workspace would avoid overflow situations during release mode.

Code Location:

increase_bond_amount function from **ref_staking** contract:

Listing 35: contracts/dexter_generator/ref_staking/src/contract.rs (Lines 188,189)

```
187 fn increase_bond_amount(state: &mut State, staker_info: &mut
    ↳ StakerInfo, amount: Uint128) {
188     state.total_bond_amount += amount;
189     staker_info.bond_amount += amount;
190 }
```

compute_staker_reward function from **ref_staking** contract:

Listing 36: contracts/dexter_generator/ref_staking/src/contract.rs (Line 235)

```
230 fn compute_staker_reward(state: &State, staker_info: &mut
    ↳ StakerInfo) -> StdResult<()> {
231     let pending_reward = (staker_info.bond_amount * state.
    ↳ global_reward_index)
232         .checked_sub(staker_info.bond_amount * staker_info.
    ↳ reward_index)?;
233
```

```
    staker_info.reward_index = state.global_reward_index;
    staker_info.pending_reward += pending_reward;
    Ok(())
}
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to use checked mathematical operations in the code to avoid future overflow/underflow situations.

Remediation Plan:

SOLVED: The Dexter team has solved this issue in commit 830cbce.

3.25 (HAL-25) IMPROPER ERROR HANDLING (POOLS) - INFORMATIONAL

Description:

In the assert_max_spread function of the **xyk_pool** and **stable_pool** contract, the value of belief_price could cause a panic if it equals zero.

As a good practice, it is recommended to avoid all possible situations where the code could panic because the panic error sometimes does not provide useful information about the root cause. For that reason, there are some scenarios where a simple pre-check could avoid the panic situation and would give a clearer message to the user/developer about what went wrong.

Code Location:

Fragment of assert_max_spread function from **xyk_pool** contract:

Listing 37: contracts/pools/xyk_pool/src/contract.rs (Line 867)

```
866 if let Some(belief_price) = belief_price {
867     let expected_return = offer_amount * belief_price.inv().unwrap();
868     let spread_amount = expected_return
869         .checked_sub(return_amount)
870         .unwrap_or(Uint128::zero());
871     let calc_spread = Decimal::from_ratio(spread_amount, expected_return);
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to perform a non-zero check on the belief_price variable at the beginning of the function, or replace unwrap() with another type or error handling function.

Remediation Plan:

SOLVED: The Dexter team has solved this issue, since the **xyk_pool** and **stable_pool** contracts have been removed from the final version of the code.

3.26 (HAL-26) LACK OF STORAGE CLEAN UP - INFORMATIONAL

Description:

multi_staking contract stores some states related to **users info** such as ASSET_STAKER_INFO, USER_LP_TOKEN_LOCKS and USER_BONDED_LP_TOKENS. These states are temporary in nature, as they are useful as long as a user is using the service.

As a good practice, once the user has finished using the service (no more assets have been bonded, already unbonded assets have been unlocked, etc.), these records should be deleted from storage to gain some efficiency in future searches and to have clear records of the current usage of the service.

Code Location:

User states from **multi_staking** contract:

Listing 38: contracts/multi_staking/src/state.rs (Lines 17,24,27)

```
16 /// user's last reward index.
17 pub const ASSET_STAKER_INFO: Map<(&Addr, &Addr, &str),
   ↳ AssetStakerInfo> = Map::new("asset_staker_info");
18
19 /// Store of all reward schedules for a (LP token, Asset) pair.
20 pub const REWARD_SCHEDULES: Map<(&Addr, &str) , Vec<RewardSchedule
   ↳ >> = Map::new("rewards");
21
22 /// This is used to keep track of the LP tokens that are currently
   ↳ locked for the user
23 /// after they have unbonded their tokens.
24 pub const USER_LP_TOKEN_LOCKS: Map<(&Addr, &Addr), Vec<TokenLock>>
   ↳ = Map::new("user_lp_token_unlocks");
25
26 /// This is used to keep track of the LP tokens that are currently
   ↳ bonded by the user.
```

```
pub const USER_BONDED_LP_TOKENS: Map<(&Addr, &Addr), Uint128> = Map::new("user_bonded_lp_tokens");
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to have a function to delete those user-related states that will no longer be used.

Remediation Plan:

ACKNOWLEDGED: The Dexter team acknowledged this finding.

THANK YOU FOR CHOOSING

// HALBORN