Audit Report

Prism

v1.0

January 20, 2022

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 Terraform Labs to perform a security audit of the Prism smart contracts.

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 behaviour.
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/prism-finance/prism-contracts

Commit hash: `198dc4834e03fc10c26f48a6d42b412a176088cd`

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

Prism allows refracting digital assets into yield and principal components. The audited smart contracts contain the core functionality as well as staking, governance, launch and periphery contracts.

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 improve 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 a security audit and vice versa.

Summary of Findings

No	Description	Severity	Status
1	Users that unbond from yAsset contract in XPrism mode receive twice their bonded amount minus the fees, and fees are not collected	Critical	Resolved
2	Lack of tax dedication in limit order contract leads to other users' funds being spent	Major	Resolved
3	Missing total bond amount decrease during unbonding from launch pool contract leaves rewards stuck in contract	Major	Resolved
4	Users might lose their bonds if unbonding periods of vault contract and Terra protocol are not the same duration	Minor	Acknowledged
5	Exchange rate peg mechanism allows users to profit from slashes and inhibits liquidity	Minor	Partially Resolved
6	Withdrawal of funds may fail if a user unbonds many times without withdrawing	Minor	Resolved
7	Protection against mass withdrawals is weak and can be bypassed with multiple accounts	Minor	Resolved
8	Lack of configuration validation in vault and yAsset staking contracts can cause errors	Minor	Resolved
9	Configuration updates in gov contract might affect votes in progress	Minor	Resolved
10	Contract updates in vault contract can corrupt shared state	Minor	Resolved
11	Reading and removing validators might fail if list of validators gets too large	Minor	Resolved
12	Query of withdrawable unbonded amount might return wrong value	Minor	Resolved
13	Whitelist lookup gets more expensive with number of whitelisted assets	Minor	Partially Resolved
14	Rewards can be fully arbitraged if attacker can time them	Minor	Resolved
15	Lack of tax deduction in yAsset staking contract	Minor	Resolved

16	Zero executor fees are sent in limit order contract, causing an error	Minor	Resolved
17	Claiming rewards will fail if PRISM token balance of LP staking contract is insufficient	Minor	Acknowledged
18	Withdrawing assets might fail if PRISM token balance of launch pool contract is insufficient	Minor	Acknowledged
19	Faulty launch config can be supplied to fair launch contract	Minor	Resolved
20	Attribute in fair launch contract on withdrawal returns wrong withdrawal fee	Minor	Resolved
21	Withdrawing voting tokens from governance contract may run out of gas	Minor	Resolved
22	Polls that are passed but cannot be executed due to the message gas limit might become executable in an unforeseeable future	Minor	Resolved
23	Inconsistent time calculation in governance contract might lead to wrong query responses	Minor	Resolved
24	Poll deposits will be stuck in the governance contract if quorum is not reached	Minor	Resolved
25	Rewards can be fully arbitraged if attacker can time them	Informational	Acknowledged
26	Delegations may be sent to inactive validators	Informational	Acknowledged
27	Users are subject to slashing between unbonding and undelegation batch execution, which is currently not documented	Informational	Resolved
28	Undelegation logic leads to low diversification of bonded tokens across validators	Informational	Acknowledged
29	Approve and withdraw pattern is inefficient	Informational	Acknowledged
30	Lack of setting expiry on allowances is bad practice	Informational	Acknowledged
31	Overflow checks not enabled for release profile in some packages	Informational	Resolved
32	Canonical address transformations are inefficient	Informational	Acknowledged

Code Quality Criteria

Criteria	Status	Comment
Code complexity	**Medium**	-
Code readability and clarity	**Medium-High**	-
Level of documentation	**Low-Medium**	The code has minimal documentation. The whitepaper outlines the high-level functionality well but does not go into much detail about the underlying mechanism. Documentation on periphery contracts is missing.
Test coverage	**High**	-

Detailed Findings

1. Users that unbond from yAsset contract in XPrism mode receive twice their bonded amount minus the fees, and fees are not collected

Severity: Critical

In `contracts/prism-yasset-staking/src/staking.rs:96-108` and `116` if `mode == StakingMode::XPrism`, the user receives `2*unbonded_amount - withdrawal_fee`. At the same time, the fees are not sent to any other contract. This leads to draining funds from other users and fees not being distributed to stakers, compromising incentives to stake.

Recommendation

We recommend setting a fee collector contract as the recipient in line `101` and using `fee` as the amount in line `102`.

Status: Resolved

2. Lack of tax deduction in limit order contract leads to other users' funds being spent

Severity: Major

In `contracts/prism-limit-order/src/order.rs:319` and `322` no taxes are deducted during the execution of the swap.

Without deducting taxes, the contract will pay taxes for the swap from its own balance, which will consume the offered amounts of other limit orders, leading to a failure to execute those other limit orders in the future. This would affect a wide range of users of the protocol, who would permanently lose their funds.

Recommendation

We recommend adding tax deduction to prevent losing funds in the relevant lines.

Status: Resolved

3. Missing total bond amount decrease during unbonding from launch pool contract leaves rewards stuck in contract

Severity: Major

In the `unbond` function in `contracts/prism-launch-pool/src/contract.rs:206-215`, the `total_bond_amount` is not decreased by the unbonded `amount`. This implies that for any further rewards, the denominator in line `277` will be too big, leading to any bonded users proportionally losing out on further rewards.

Recommendation

We recommend decreasing `total_bond_amount` by `amount` as done in the procedure for bonding in lines `187-191`.

Status: Resolved

4. Users might lose their bonds if unbonding periods of vault contract and Terra protocol are not the same duration

Severity: Minor

If no batch is released in `contracts/prism-vault/src/unbond.rs:255`, the actual unbonded amount is still set to zero in line `317`. Moreover, the `prev_vault_balance` is updated in the calling function `execute_withdraw_unbonded` in line `191`. This might cause lost funds because the vault unbonding period can be freely set, while the Terra unbonding period is fixed to 21 days.

Two cases are possible. On the one hand, if the unbonding period used in the contract is longer than the actual unbonding period of the Terra protocol, this might lead to lost funds. On the other hand, if the unbonding period is set to less than the actual unbonding period, slashing of validators might lead to a situation where users can avoid being affected by slashing at the cost of other users because the slashing is not executed.

Despite the severe implications of a mismatch of the unbonding periods, we classify this issue as minor, since governance can ensure equality of the periods by carefully making required changes.

Recommendation

We recommend changing the `execute_withdraw_unbonded` function to gracefully handle a longer unbonding period by only setting the `actual_unbonded_amount` in line `317` to zero and updating the `prev_vault_balance` in line `191` if batches have been processed, and ensuring the contract's unbonding period is at least as long as the one of the Terra protocol.

This recommendation does not address the case where the unbonding period of Prism is shorter than the unbonding period of Terra. To fully address the issue, a query that returns Terra's unbonding period would be necessary.

Status: Acknowledged

The Prism team acknowledges this issue, stating that they will take care of the appropriate governance actions in the unlikely case that the Terra unbonding period changes.

5. Exchange rate peg mechanism allows users to profit from slashes and inhibits liquidity

Severity: Minor

The exchange rate peg mechanism implemented in `contracts/prism-vault/src/bond.rs:119-129` and `contracts/prism-vault/src/unbond.rs:41-51` might compromise the incentives of users and validators, depending on the parameter values of the `threshold` and the `recovery_fee`. There might be a free lunch for any user that bonds directly after a possible slashing event.

First, this compromises computations made in `contracts/prism-yasset-staking/src/rewards.rs:28-35` where the `stakers_portion` could become larger than one.

Second, in the worst case, if a potential profit from a free lunch is large enough, validators might be incentivised to trigger slashing in order to realize these profit opportunities. This might compromise the protocol.

Third, users that wish to unbond after slashing have to pay up to the maximal peg recovery fee `peg_fee` in order to restore the peg. This leads to a first-mover disadvantage in the sense that one single user might need to pay for the whole slashing. This in turn disincentivizes unbonding and reduces the liquidity of all derivative tokens.

These issues are caused by the peg recovery mechanism. In contrast to the Anchor protocol, which uses a similar mechanism, Prism does not have a natural peg recovery by accruing rewards on cLUNA.

Recommendation

We recommend refining the exchange rate model and peg recovery mechanism. Slashing should affect all users that have bonded equivalently and new bonding users should not profit from recent slashing.

Status: Partially Resolved

The Prism team fixed the first point and the team acknowledges this issue, stating that they and their community will carefully select validators to prevent malicious actors. In addition the team states that the parameter values of `threshold` and the `recovery_fee` will be 1 and smaller than 1%, respectively, to prevent compromised incentives.

6. Withdrawal of funds may fail if a user unbonds many times without withdrawing

Severity: Minor

In `contracts/prism-vault/src/state.rs:60`, `76`, `89` and `114`, iterations over the `UNBOND_WAITLIST` may run out of gas, if the list has many entries. Many entries might occur if a user unbonds regularly, but does not withdraw. That could happen for example if the user is a bot that automatically bonds daily. As a consequence, withdrawing funds may fail. There is currently no way to recover from this issue.

Similarly the query in line `139` can run out of gas and fail.

Recommendation

We recommend allowing the specification of a limit in `execute_withdraw_unbonded` in `contracts/prism-vault/src/unbond.rs:151` that is applied to the `get_finished_amount` and `get_unbond_batches` functions, but not to the `process_withdraw_rate` function.

Status: Resolved

7. Protection against mass withdrawals is weak and can be bypassed with multiple accounts

Severity: Minor

In `contracts/prism-fair-launch/src/contract.rs:185-187` in order “to prevent massive withdraw on phase 2” (as stated in the comment), a fee is charged if `withdraw_amount` is greater than `cfg.withdraw_threshold`. This mechanism can easily be avoided by withdrawing multiple times with amounts less than or equal to `cfg.withdraw_threshold`. Elaborate exploiters can also bypass it easily by using multiple accounts to execute greater withdrawals.

Recommendation

We recommend refining the conditions and computations of the withdrawal fee. Likely multiple conditions will be necessary to soundly protect against mass withdrawals, while maintaining high incentives to commit to the launch pool. These will include conditions related to the absolute and relative amount of the withdrawn amount in relation to both the total deposited value by all accounts and the individual deposited value. In addition, a withdrawal fee that increases towards the end of the phase 2 can prevent a massive last minute withdrawal. Alternatively, using a price discovery mechanism for the withdrawal fee might be worth considering. At a minimum, the withdrawal fee should be applied to the aggregated withdrawal amount to prevent a split of the total withdrawal amount into multiple smaller ones. Another approach is to limit the amount of withdrawals to one per account.

Status: Resolved

Comment: Now in phase 2 users can withdraw only one time, and the withdrawable amount decreases over time every hour from 100% to 0%.

8. Lack of configuration validation in vault and yAsset staking contracts can cause errors

Severity: Minor

The configuration of `peg_recovery_fee` and `er_threshold` can be set to values larger than one in `contracts/prism-vault/src/contract.rs:78-79` as well as in `contracts/prism-vault/src/config.rs:38-39`. This might cause errors in `contracts/prism-vault/src/bond.rs:124-126` and `contracts/prism-vault/src/unbond.rs:45-47`.

An equivalent problem occurs in `prism-yasset-staking/src/contract.rs:37` and `42` where `protocol_fee` and `withdraw_fee` can be set to values larger than one, which might cause errors in `contracts/prism-yasset-staking/src/rewards.rs:70` and `prism-yasset-staking/src/contract.rs:116`, respectively.

We consider this a minor issue, since it can only be caused by oversight of the contract owner. However, errors in production can lead to downtime of the protocol.

Recommendation

We recommend validating configuration values to ensure no errors and panics can occur.

Status: Resolved

9. Configuration updates in gov contract might affect votes in progress

Severity: Minor

In `contracts/prism-gov/src/contract.rs:205-210`, `quorum` and `threshold` can be updated. This affects votes in progress in `contracts/prism-gov/src/polls.rs:141-146`, which could affect the outcome of a vote. In addition, proposers might lose their deposit on existing polls after a change of the quorum.

As the values can only be updated by the contract owner or through governance, we consider this a minor issue. An update coming through governance would be visible by any proposer at the time of poll creation. Since the outcome of a poll that updates `quorum` or `threshold` values might not be certain, proposers could be disincentivized to create new polls during such a period.

Recommendation

We recommend that `quorum` and `threshold` are stored within each poll and get assigned the value of the current configuration.

Status: Resolved

10.Contract updates in vault contract can corrupt shared state

Severity: Minor

Changing any of the contracts in `contracts/prism-vault/src/config.rs:75-113` without state migration may leave the contracts corrupted. Since the values can only be updated by the creator or governance we consider this a minor issue. However, errors in production can lead to downtime of the protocol.

Recommendation

We recommend that the contracts can only be set once.

Status: Resolved

11. Reading and removing validators might fail if list of validators gets too large

Severity: Minor

In `contracts/prism-vault/src/state.rs:171`, `read_validators` might run out of gas if `VALIDATORS` has too many entries. This compromises the functionality of the protocol as validators might not be removable anymore because of the call in `contracts/prism-vault/src/config.rs:171`. Also bonding without specifying a validator might fail due to `contracts/prism-vault/src/bond.rs:73`.

Recommendation

We recommend setting and enforcing a maximum number of validators.

Status: Resolved

12. Query of withdrawable unbonded amount might return wrong value

Severity: Minor

In the `query_withdrawable_unbonded` function in `contracts/prism-vault/src/contract.rs:478`, `all_requests` might not be equal to the actual withdrawable amount, because the withdrawal rate might not have been processed as in `contracts/prism-vault/src/unbond.rs:173`. Consequently, potential slashing might not have been accounted for.

Recommendation

We recommend calling `process_withdraw_rate` in the `query_withdrawable_unbonded` function.

Status: Resolved

13. Whitelist lookup gets more expensive with number of whitelisted assets

Severity: Minor

Whitelisted assets are stored as a vector in `WHITELISTED_ASSETS`, which implies that adding more assets through `contracts/prism-yasset-staking/src/rewards.rs:256` will make any function accessing this list more expensive. Since the list is unbounded, functions accessing it could eventually even run out of gas. There is currently no way to recover from that.

Recommendation

We recommend replacing the data structure with a map such that gas consumption of other functions is independent of the number of whitelisted assets. Alternatively, an upper limit for the number of whitelisted assets could be set, together with a function to remove whitelisted assets.

Status: Partially Resolved

The Prism team acknowledges this issue, stating that they will not whitelist more than about 10 assets. A function to remove whitelisted assets has been added to prevent unlimited growth of the list.

14. Lack of tax deduction in yAsset staking contract

Severity: Minor

In `contracts/prism-yasset-staking/src/swaps.rs:71`, no taxes are deducted during the execution of the swap.

Without deducting taxes, the contract will pay taxes for the swap from its own balance. This is only a minor issue because the LUNA tax rate is currently 0%. But as this might change in the future and `reward_denom` can be set to any denomination, it might compromise the functionality in the future.

Recommendation

We recommend deducting taxes.

Status: Resolved

15. Zero executor fees are sent in limit order contract, causing an error

Severity: Minor

In `contracts/prism-limit-order/src/order.rs:268`, it is not checked if `executor_fee_asset` is greater than zero, as is done in line `281`. If the amount is zero, the message will fail, reverting execution of the order.

Recommendation

We recommend not sending funds if `executor_fee_asset` is zero.

Status: Resolved

16. Faulty distribution schedule config can cause panics in LP staking contract

Severity: Minor

In `contracts/prism-lp-staking/src/contract.rs:24` a `distribution_schedule` may be stored with a start time that is greater than the end time. This would cause `contracts/prism-lp-staking/src/handle.rs:247` to panic.

For informational purposes, we also highlight that it is possible to supply a schedule with gaps and overlaps. Such gaps and overlaps do not cause errors, but might not be intended.

Recommendation

We recommend adding validation to ensure the end of a distribution schedule is greater than the start.

Status: Resolved

17. Claiming rewards will fail if PRISM token balance of LP staking contract is insufficient

Severity: Minor

In `contracts/prism-lp-staking/src/handle.rs:209-218` claiming rewards might fail if the PRISM token balance controlled by the contract is not large enough.

Recommendation

We recommend changing the setting of the distribution schedule to a CW20 receive message that validates that the PRISM amount set corresponds to the sum of distributed tokens in the distribution schedule, or minting tokens when distributing them.

Status: Acknowledged

18. Withdrawing assets might fail if PRISM token balance of launch pool contract is insufficient

Severity: Minor

In `contracts/prism-launch-pool/src/vest.rs:93-99` withdrawing might fail if the PRISM token balance controlled by the contract is not large enough.

Recommendation

We recommend changing the setting of the distribution schedule to a CW20 receive message that validates that the PRISM amount set corresponds to the sum of distributed tokens in the distribution schedule, or minting tokens when distributing them.

Status: Acknowledged

19. Faulty launch config can be supplied to fair launch contract

Severity: Minor

In `contracts/prism-fair-launch/src/contract.rs:73`, it is checked whether `launch_config.phase2_end > launch_config.phase2_end`, which will always be false. Therefore, it is possible to supply a launch config that allows for phase 2 starting after the end of phase 2.

Since the values can only be updated by the creator or governance we consider this a minor issue. However, it would compromise the whole incentive structure of the launch-pool.

Recommendation

We recommend replacing line `73` by `launch_config.phase2_start > launch_config.phase2_end`.

Status: Resolved

20. Attribute in fair launch contract on withdrawal returns wrong withdrawal fee

Severity: Minor

In `contracts/prism-fair-launch/src/contract.rs:191` the `withdraw_fee_amount` gets assigned the value of the full `withdraw_amount` and is then reported wrongly in the `withdraw_fee` attribute in line `203`.

Recommendation

We recommend replacing `withdraw_amount` with `withdraw_fee_amount` in line `191`.

Status: Resolved

21. Withdrawing voting tokens from governance contract may run out of gas

Severity: Minor

In `contracts/prism-gov/src/voting.rs:81-97` iterations over the `locked_balance` vector in the `token_manager` are unbounded. These iterations could run out of gas if a user voted on too many polls. As a consequence, withdrawing voting tokens can fail.

We classify this as minor because it is not likely that a user voted on so many polls.

Recommendation

We recommend setting and enforcing a maximum number of polls a user can vote on.

Status: Resolved

22. Polls that are passed but cannot be executed due to the message gas limit might become executable in an unforeseeable future

Severity: Minor

In `contracts/prism-gov/src/polls.rs:211-220`, messages could reach the message gas limit, which would lead to a revert of the whole transaction. In such a case, the reply that is supposed to run on error will not run, leaving the poll in the passed state forever, with no way to change the status. If the Terra blockchain ever raises the message gas limit, such an old poll might become executable, but may long not be relevant anymore or even harmful. This could lead to unintended state changes in the far future.

Recommendation

We recommend adding an expiry time to polls, such that if a poll is not executed past an `expiry_period`, it cannot be executed anymore.

Status: Resolved

The Prism team decided to resolve this issue by setting a gas limit to the message sent during poll execution.

23. Inconsistent time calculation in governance contract might lead to wrong query responses

Severity: Minor

In `contracts/prism-gov/src/xprism.rs:192` the condition that checks whether the current time is greater than the `end_time` is inconsistent with the condition in line `144` that checks whether the current time is greater than or equal to the `end_time`. This might lead to a wrong query result of the `claimable_amount`.

Recommendation

We recommend replacing > with >= in line `192`.

Status: Resolved

24. Poll deposits will be stuck in the governance contract if quorum is not reached

Severity: Minor

If the quorum is not reached In `contracts/prism-gov/src/polls.rs:141`, the deposit will remain in the governance contract. This effectively decreases the xPRISM token supply, which is a benefit to all xPRISM holders. Locked instead of burned tokens do not show up properly in most user interfaces though.

Recommendation

We recommend either explicitly burning the xPRISM tokens of polls that did not reach the quorum, or distributing those tokens explicitly.

Status: Resolved

25. Rewards can be fully arbitraged if attacker can time them

Severity: Informational

In `contracts/prism-yasset-staking/src/rewards.rs:84`, the `pool_info.reward_index` is updated when rewards are deposited. When an attacker can time the `DepositRewards` message (i.e. know when it is executed or execute them themselves), they can bond directly before and unbond directly after it to arbitrage all the rewards, proportional to their bonded amount. This makes yLUNA very valuable around the `DepositRewards` messages and less valuable at all other times, which may create a lot of price-volatility if it is known by all market participants or just heavily discounts yLUNA if it remains unknown to the majority of users.

This is only a minor issue since the function `deposit_rewards` in `contracts/prism-yasset-staking/src/rewards.rs:18` is permissionless and all users have some incentive to execute it regularly, which lowers the arbitrable amounts and potential volatility/discounts.

Recommendation

We recommend that the Prism team sends the `DepositRewards` message regularly, but at randomized times. Moreover, non-deposited rewards should be communicated to users actively. Alternatively and/or additionally, limits to bonding amounts or smoothing the deposited rewards over time can mitigate the issue.

Status: Acknowledged

The Prism team acknowledges this issue, stating that they will send the `DepositRewards` message multiple times per day.

26. Delegations may be sent to inactive validators

Severity: Informational

In `contracts/prism-vault/src/bond.rs:78-83` does not ensure that the picked `validator` is active. The user might delegate to an inactive/jailed/tombstoned validator.

Recommendation

We recommend mitigating this issue through off-chain means such as removing inactive validators from the whitelist and showing warnings on the frontend if a picked validator is inactive.

Status: Acknowledged

27. Users are subject to slashing between unbonding and undelegation batch execution, which is currently not documented

Severity: Informational

The current implementation performs undelegations in batches for efficiency reasons. Undelegations happen at most every `epoch_period` through the logic in `contracts/prism-vault/src/unbond.rs:76-120`.

That implies that users that have sent unbond messages to the vault contract will still be subject to slashing until the undelegation batch is executed. This behaviour is different from Terra's/Cosmos SDK's slashing module, which only slashes delegators that were active when the infraction happened.

This difference is currently not documented.

Recommendation

We recommend documenting that undelegated funds in Prism will still be subject to slashing until the undelegation is executed, which can only happen after the current epoch ends.

Status: Resolved

28. Undelegation logic leads to low diversification of bonded tokens across validators

Severity: Informational

In the current implementation, unbonding picks one or more pseudo-random validators to undelegate from in `contracts/prism-vault/src/unbond.rs:341-368`. This algorithm leads to an uneven distribution of the stake across whitelisted validators, causing potentially low diversification and higher risk.

Recommendation

We recommend evening out delegated funds when users unbond.

Status: Acknowledged

29. Approve and withdraw pattern is inefficient

Severity: Informational

In different places in the codebase, an approve and withdraw pattern is used to transfer CW20 tokens. Usage of approvals is generally considered a bad practice since they require an additional transaction from the account to grant the allowance and require care in setting appropriate limits or revoke the allowance after interaction with the contract is done. That leads to a degradation in usability. Instances of the approve and transfer pattern are found in:

- `contracts/prism-vault/src/contract.rs:377-392`
- `contracts/prism-limit-order/src/order.rs:58-66`

Recommendation

We recommend using the CW20 receive pattern instead.

Status: Acknowledged

30. Lack of setting expiry on allowances is bad practice

Severity: Informational

In multiple places in the codebase, the `expires` field of the `IncreaseAllowance` message is set to `None`. This is generally considered a bad practice, since the implementation of the allowance holder might change in the future, potentially leading to unintended consequences. Instances of this issue are:

- `contracts/prism-vault/src/contract.rs:382`
- `contracts/prism-yasset-staking/src/rewards.rs:163`

Recommendation

We recommend setting the `expires` field in all instances above to `env.block.height + 1`.

Status: Acknowledged

31. Overflow checks not enabled for release profile in some packages

Severity: Informational

The following `Cargo.toml` files do not enable overflow-checks for the release profile:

- `contracts/prism-astro-generator-proxy/Cargo.toml`
- `contracts/prism-fair-launch/Cargo.toml`
- `contracts/prism-launch-pool/Cargo.toml`
- `packages/astroport/Cargo.toml`

Recommendation

Even though this check is implicitly applied to all packages from the workspace's `Cargo.toml`, we recommend also explicitly enabling overflow checks in every individual package. That helps prevent unintended consequences when the codebase is refactored in the future.

Status: Resolved

32. Canonical address transformations are inefficient

Severity: Informational

While previously recommended as a best practice, usage of canonical addresses for storage is no longer encouraged. The background is that canonical addresses are no longer stored in a canonical format, so the transformation just adds overhead without much benefit. Additionally, the codebase is more complicated with address transformations.

Recommendation

We recommend removing any transformation from human to canonical addresses and using the new `Addr` type for validated addresses instead.

Status: Acknowledged