Audit Report

Eris Protocol

v1.0

February 15, 2023

Table of Contents

License

THIS WORK IS LICENSED UNDER A CREATIVE COMMONS ATTRIBUTION-NODERIVATIVES 4.0 INTERNATIONAL LICENSE.

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

COPYRIGHT OF THIS REPORT REMAINS WITH THE AUTHOR.

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 Eris Protocol to perform a security audit of their new liquid staking derivative for Terra, Eris Amplified Staking.

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 target:

Repository	https://github.com/erisprotocol/contracts-terra
Initial commit	`9b0744dc6a2614d0582d12131816d4a9dfd4b0a0`
Fixes verified at commit	`e22a1a6721bc62c98f472491e9cd509bc63683bf`
Scope	The scope of this audit was limited to: - `contracts/hub` - `contracts/amp-governance/voting_escrow`

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

The smart contracts within the scope of this audit implement part of the functionalities from Eris Protocol's liquid staking derivative project for Terra.

The following was in the scope of this audit:

- The `hub` contract, which allows bonding and unbonding of Luna, minting and burning of ampLUNA, and reinvestment of staking rewards.
- The `voting-escrow` contract, which enables boosting governance and delegation power to ampLP token holders through staking.
- Relevant imports from `packages`.

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**	Although the code is well architected, there is an overall lack of inline comments that describe the purpose and arguments of functions.
Level of documentation	**Medium-High**	-
Test coverage	**High**	`cargo tarpaulin` reported a test coverage of 90.27% for the `voting-escrow` and 91.86% for the `hub` contract.

Summary of Findings

No	Description	Severity	Status
1	First depositor can be front-run for unfair profit causing direct losses	Critical	Resolved
2	Underflow upon slope calculation may lead to unreliable voting power	Major	Resolved
3	`reconcile_batches` can underflow, blocking any reconciliations	Major	Resolved
4	`ExtendLockAmount` and `DepositFor` can be abused to briefly gain voting power cheaply	Major	Resolved
5	Unharvested rewards not accounted for in `compute_mint_amount` function	Minor	Acknowledged
6	Period rounding can lead to undesired results	Minor	Acknowledged
7	Lack of validation of embedded logos	Minor	Resolved
8	Lack of validation upon instantiation of the Hub contract	Minor	Resolved
9	Insufficient validation in address blacklisting function	Minor	Resolved
10	Coefficient calc implementation differs from documentation	Informational	Resolved
11	Widespread usage of generic errors in the Hub contract	Informational	Resolved
12	Lack of “drop proposal” function on owner transfer mechanism	Informational	Resolved
13	Hardcoded denom and `EPOCH_START`	Informational	Resolved
14	Optimization in `compute_undelegations`	Informational	Resolved
15	Outdated documentation	Informational	Resolved
16	Unnecessary conversion to lowercase in addresses	Informational	Resolved
17	Incorrect comments and typographical errors	Informational	Resolved
18	Commented code	Informational	Resolved

19	“Migrate only if newer” pattern not followed	**Informational**	**Acknowledged**

Detailed Findings

1. First depositor can be front-run for unfair profit causing direct losses

Severity: Critical

The hub contract does not prevent the first depositor from being front-run to effectively get fewer shares than planned from which the attacker will profit.

In `contracts/hub/src/math.rs:36-38`, the number of shares that a user receives depends on `ustake_supply`, `uluna_to_bond`, and `uluna_bonded`. However, if a malicious depositor makes a large enough "donation" to the vault at the right time, increasing `uluna_bonded`, the next depositor will not receive their expected amount of shares.

The below describes a potential exploit scenario that could be followed by an attacker "Mallory" to take advantage of a victim "Alice":

1. Upon identifying that Alice is trying to make the first deposit of `X uluna` into the `hub` contract, Malloryfront-runs their transaction with two calls:
   a. Making a minimal initial deposit, let's say `1 uluna`, to obtain one unit of shares and setting `ustake_supply` to one.
   b. Donating a number of tokens to the contract equal to `X/2 uluna`, increasing the `bonded_luna` value without increasing `ustake_supply`.
2. Alice's transaction gets executed containing the `ExecuteMsg:Bond` message, expecting to receive `X` amount of shares. As their deposit is just `1 uluna` below the amount required to get `2` shares, only one will be rewarded.
3. As a result, `bonded_luna` would be equal to `1 + 3x/2` and `ustake_supply` would be `2`. Making each share worth `1/2 + 3x/4 uluna`.
4. Mallory will then be able to return the share to get `1/2 + 3x/4 uluna` after having spent just `1 + X/2 uluna`, effectively profiting `X/4 uluna`.

Note that this issue is only exploitable at the beginning of the contract's lifecycle and does only affect the first user making the deposit. However, as the potential loss of funds can be substantial we classify it as critical.

Recommendation

We recommend performing the first deposit atomically upon contract deployment. Their first deposit should be large enough to limit the impact of the described exploit.

Status: Resolved

2. Underflow upon slope calculation may lead to unreliable voting power

Severity: Major

The `voting_escrow` contract calculations of slope increments were affected by integer underflow in `contracts/amp-governance/voting_escrow/src/utils.rs:80`, `contracts/amp-governance/voting_escrow/src/contract.rs:268`, and `282`, leading to incorrect voting power calculations in line `52` of `utils.rs`.

Although the potential impact of this issue is critical, line `52` makes use of `checked_sub` which limits the impact. Still, this causes dangerous inconsistencies that could lead to a wide variety of unexpected ramifications.

This issue was found by the Eris team during the audit.

Recommendation

We recommend making use of `saturating_sub` in the affected arithmetic operations.

Status: Resolved

In addition, the Eris team stated they will monitor potential inconsistencies.

3. `reconcile_batches` can underflow, blocking reconciliations

Severity: Major

The `hub` contract's `reconcile_batches` function subtracts `uluna_for_batch` from `batch.uluna_unclaimed` in `contract/hub/src/math.rs:348`. However, it can happen that one batch has a very low value for `batch.uluna_unclaimed` when there were only a few or no contributions in place. In such a situation, this subtraction will underflow and cause a panic, preventing any further reconciliations.

The severity of this issue has been set to major as the situation is recoverable by adding `uluna` to the smart contract, avoiding the rest of the funds getting locked forever.

A proof of concept unit test has been crafted for this issue and can be found in the Appendix Test case for reconcile_batches can underflow, blocking any progress.

Recommendation

We recommend performing a saturating subtraction and subtracting the difference from other batches if the full amount can not be deducted from the current batch.

Status: Resolved

4. `ExtendLockAmount` and `DepositFor` can be abused to briefly gain voting power cheaply

Severity: Major

The `voting-escrow` contract calls the function `deposit_for` when the `Cw20HookMsg::ExtendLockAmount` or `Cw20HookMsg::DepositFor` hook messages are received. This function only checks that the lock has not expired yet, but it does not check how much time is left – unlike `create_lock`, which enforces `MIN_LOCK_PERIODS`.

Users can lock up capital for a very short amount of time to increase their voting power significantly. For instance, a user may use `Cw20HookMsg::ExtendLockAmount` one minute before an existing lock expires to increase their voting power by taking a very short-term loan, getting voting power for a very small amount of fees.

Recommendation

We recommend enforcing `MIN_LOCK_PERIODS` also in `deposit_for` to disallow increasing the locked amount when the lock is soon to be expired.

Status: Resolved

5. Unharvested rewards not accounted for in `compute_mint_amount` function

Severity: Minor

The `compute_mint_amount` function of the `hub` contract, located at `contracts/hub/src/execute.rs:144`, does not take into account unharvested rewards when calculating the mint amount for a new user. As the protocol reinvests rewards, these belong to previous users, and new users are also given a share of these new rewards.

Protocol users have a natural incentive to call the `harvest` function to minimize any unharvested rewards getting distributed to other users. We classify this issue as minor, since the client states it is expected behaviour and that they will explain this in the user interface.

Recommendation

We recommend:

1. Calling the `harvest` function before computing the new shares in the `compute_mint_amount` function.
2. Incentivizing users to call the `harvest` function more frequently in order to reduce the impact of the issue.

Status: Acknowledged

The client stated that this would lead to high gas fees for each deposit. As they are harvesting daily and the unbonding period is 21 days, users can get a maximum of 22 days of rewards for 21 days or 4.7% of rewards extra. Further, it is planned to increase the harvesting interval to reduce this amount even further.

6. Period rounding can lead to undesired results

Severity: Minor

The function `create_lock` converts a timestamp into a number of periods to get the current block's period (variable `block_period`) and to calculate the number of periods for the lock (variable `periods`). In both of these operations, rounding occurs which can lead to unexpected and potentially undesired results.

The rounding down of `block_period` means that the start of the lock could be set to the current period. Because of that, the minimum number of periods will not be strictly enforced. For instance, if a user creates a lock just before a period ends, the capital is locked up for one week less than if he creates a lock just after a period starts.

On the other hand, the rounding down of `periods` may confuse a user. If a user specifies a `time` that corresponds to 3.99 weeks, the lock will only be created for three periods (three weeks).

Recommendation

We recommend:

- For `block_period`, consider rounding up or incorporating the fraction (after rounding down) into the calculation for the time limit.
- For `periods`, consider rounding to the nearest integer, which should be less confusing (as a provided time of 3.99 weeks would result in a lock period of four weeks).

Status: Acknowledged

The client stated that in the GUI the user can only select locks based on period. The current period is always the start of the lock. It is also acknowledged that "3 weeks min lock" means that a user can lock their funds for 2 weeks + 1 second in the best case.

7. Lack of validation of embedded logos

Severity: Minor

The `voting-escrow` contract does not validate embedded logos during its `validate_marketing_info` function in `contracts/amp-governance/voting_escrow/src/marketing_validation.rs:52-69`.

If an embedded logo is submitted on instantiation it will be saved without any further check. This is not the case for the `execute_upload_logo` function of CW20, which successfully validates all cases through `validate_logo`.

Recommendation

We recommend adding an additional case on the `validate_marketing_info` function to check embedded logos using the `cw20::verify_logo` function.

Status: Resolved

8. Lack of validation upon instantiation of the Hub contract

Severity: Minor

The `hub` contract's instantiation function in `contracts/hub/src/execute.rs:33` lacks some validation steps of `epoch_period` and `unbond_period`. In addition, the list of validators stored in line `49` is checked for duplicates but it does not actually validate addresses to be those of active validators. The same situation is found in the `add_validator` function in lines `666-682`.

We classify this issue as minor since instantiation parameters are usually reviewed with care by a privileged user or development team before deployment.

Recommendation

Validation on `epoch_period` and `unbond_period` should at least ensure that those parameters are not zero, as it will render some of the functionalities unusable. Additionally, we consider it a good practice to enforce more detailed bounds by defining a range of expected values for normal usage.

We recommend checking that the supplied validator addresses are actually those of existing validators through the `StakingQuery::Validator` message.

Status: Resolved

9. Insufficient validation in address blacklisting function

Severity: Minor

The `voting-escrow` contract's `update_blacklist` function checks that the two vectors provided as arguments, `append_addrs` and `remove_addrs`, do not clash with the current contents of the blacklist in `contracts/amp-governance/voting_escrow/src/contract.rs:707-714`. However, it does not check if addresses are duplicated within each vector or if an address is part of both at the same time.

Duplicated addresses in `append_addrs` cause minor inefficiencies as those will be saved into the storage, unnecessarily growing it with no purpose and opening a window for future inconsistencies. This limited impact is mitigated by the `if` statement in line `744`, as processing the first instance of any address causes the `cur_power` of duplicated iterations to be zero.

On the other hand, duplicates in `remove_addrs` have a real impact as there are no checks in place to avoid repeated calls to the `checkpoint` function in line `778`. This leads to additional accounting in voting power in line `347` linked to the blacklisting of the same address, compromising the integrity of the scheduled slope changes and the stored `Point`.

For addresses that are present in both vectors, the function first appends addresses and then removes them. If, by mistake, the same address is part of both, that address will not be contained in the final blacklist without the owner or guardian being aware of this situation.

Although the impact of this issue can be considered major, the function is restricted to privileged addresses which are less prone to error in general. Therefore, this issue has been raised as minor.

Recommendation

We recommend validating that each address is unique within the `append_addrs` and `remove_addrs` vectors and checking for duplicates between them.

Status: Resolved

10. Coefficient calc implementation differs from documentation

Severity: Informational

The `calc_coefficient` function found in `packages/eris/src/helpers/slope.rs:6-10` is documented with the following line: `` Coefficient calculation where 0 [`WEEK`] is equal to 1 and [`MAX_LOCK_TIME`] is 9 ``. However, the actual implementation does not adhere to this comment as when the `interval` is zero weeks, the returned value will be zero instead of one.

This issue has been raised as informational as every instance that calls the affected function checks the supplied `interval` to not be zero before or can't be zero given the contrac's logic.

Recommendation

We recommend modifying the function's implementation to reflect the documentation comment.

Status: Resolved

11. Widespread usage of generic errors in the Hub contract

Severity: Informational

The error handling in the `hub` contract is based on `StdError::generic_err` instead of declaring custom errors, as opposed to the `voting-escrow` contract which mostly relies on custom errors.

Although not a security issue, defining individual generic error messages decreases maintainability.

Recommendation

We recommend declaring custom errors and using those consistently throughout the codebase.

Status: Resolved

12. Lack of "drop proposal" function on owner transfer mechanism

Severity: Informational

The `hub` contract implements a two-step transfer mechanism for updating the `owner` address, following best practices. However, in order to remove a `new_owner` address submitted by mistake, the current owner would have to call `transfer_ownership` again and overwrite the address.

Although effective in practice, it makes more sense to have an entry point with the sole purpose of clearing this value.

Recommendation

We recommend adding a new entry point and function to the two-step transfer feature that allows the current owner to clear the value of `state.new_owner`.

Status: Resolved

13. Hardcoded denom and `EPOCH_START`

Severity: Informational

The `hub` contract makes use of the `CONTRACT_DENOM` constant to set the denom in multiple instances, following best practices. However, the following instances have been found to use a hardcoded `uluna` string instead of the `CONTRACT_DENOM` constant:

- `contracts/hub/src/types/staking.rs:44,67,93`
- `contracts/hub/src/queries.rs:80`

Although not a security issue, it impacts maintainability and may lead to errors in the future, such as a partial change of denoms throughout the codebase.

In addition, the code in `packages/eris/src/governance_helper.rs:16` defines a variable `EPOCH_START` with a specific value assigned to it. It is important to make sure that this value is correct and intended for the specific use case before deploying the code to production, as it may need to be adjusted or replaced with a dynamic value that can change over time.

Recommendation

We recommend substituting the hardcoded denom values with the `CONTRACT_DENOM` constant. In addition, the `EPOCH_START` value should be reviewed.

Status: Resolved

The client states that the `EPOCH_START` value will be constant forever and used throughout all different Eris contracts with the same value.

14. Optimization in `compute_undelegations`

Severity: Informational

The `hub` contract uses a loop that iterates over `merge_with_validators` in `contracts/hub/src/math.rs:86`. However, if the variable `uluna_to_undelegate` is set to zero, the subsequent computations on lines `91` and beyond will have no effect, wasting computational resources.

Recommendation

We recommend including a check for the value of `uluna_to_undelegate` and executing the loop only if the value is non-zero.

Status: Resolved

15. Outdated documentation

Severity: Informational

The README file in the `hub` folder mentions a message type `ExecuteMsg::SubmitUnbond`, which does not exist and was, presumably, replaced by `ExecuteMsg::SubmitBatch`.

Recommendation

We recommend updating the documentation.

Status: Resolved

16. Unnecessary conversion to lowercase in addresses

Severity: Informational

The contracts within scope used the `addr_validate_to_lower` helper function to sanitize addresses, found at `packages/eris/src/helper.rs:25-31`. Since `CosmWasm 1.0.0`, the `addr_validate` utility also validates address capitalization, hence making it redundant to perform this check manually.

Recommendation

We recommend removing the `addr_validate_to_lower` function and performing address validation through `api.addr_validate` instead.

Status: Resolved

17. Incorrect comments and typographical errors

Severity: Informational

The following incorrect comments and typographical errors were found in the audited codebase. Although not security related, they decrease documentation quality and readability of the codebase:

- `contracts/amp-governance/voting_escrow/src/contract.rs:724` "accumulator for old slopes" instead of "accumulator for old amount".
- `contracts/amp-governance/voting_escrow/src/state.rs:17` "list of whitelisted logo urls prefixes" instead of "list of contracts to receive updates on user's lock info", or something similar.
- `contracts/amp-governance/voting_escrow/src/state.rs:44` "the timestamp when the lock position starts" instead of "the period when the lock position starts"
- `contracts/hub/src/contract.rs:141` "must be 1-2" instead of "must be 1"

- `contracts/hub/src/execute.rs:305` and `309 validatiors` instead of `validators`

Recommendation

We recommend correcting these incorrect comments and typographical errors.

Status: Resolved

18. Commented code

Severity: Informational

A commented function was found in `packages/eris/src/helpers/slope.rs:20-24`. Although not a security issue, commented code decreases readability and may cause issues if re-introduced in the future without being properly re-architected.

Recommendation

We recommend removing commented code.

Status: Resolved

19.“Migrate only if newer” pattern not followed

Severity: Informational

The contracts within scope are currently migrated in an asynchronous fashion. This is fine, but the pattern can be improved by adding validation to ensure that the migration is only occurring if the version supplied is newer.

Recommendation

We recommend following the migrate “only if newer” pattern defined in the CosmWasm documentation is recommended.

Status: Acknowledged

Appendix: Test Cases

1. **Test case for `reconcile_batches` can underflow, blocking any progress**

```rust
#[test]
fn reconciling_underflow() {
   let mut deps = setup_test();
   let state = State::default();

   let previous_batches = vec![
       Batch {
           id: 1,
           reconciled: true,
           total_shares: Uint128::new(92876),
           uluna_unclaimed: Uint128::new(95197), // 1.025 Token per Stake
           est_unbond_end_time: 10000,
       },
       Batch {
           id: 2,
           reconciled: false,
           total_shares: Uint128::new(1345),
           uluna_unclaimed: Uint128::new(1385), // 1.030 Token per Stake
           est_unbond_end_time: 20000,
       },
       Batch {
           id: 3,
           reconciled: false,
           total_shares: Uint128::new(1456),
           uluna_unclaimed: Uint128::new(1506), // 1.035 Token per Stake
           est_unbond_end_time: 30000,
       },
       Batch {
           id: 4,
           reconciled: false,
           total_shares: Uint128::new(1),
           uluna_unclaimed: Uint128::new(1),
           est_unbond_end_time: 30001,
       },
   ];

   for previous_batch in &previous_batches {
       state
           .previous_batches
           .save(deps.as_mut().storage, previous_batch.id, previous_batch)
           .unwrap();
   }
```

```
    state
        .unlocked_coins
        .save(
            deps.as_mut().storage,
            &vec![
                Coin::new(10000, CONTRACT_DENOM),
                Coin::new(234, "ukrw"),
                Coin::new(345, "uusd"),
                Coin::new(
                    69420,

"ibc/0471F1C4E7AFD3F07702BEF6DC365268D64570F7C1FDC98EA6098DD6DE59817B",
                ),
            ],
        )
        .unwrap();

    deps.querier.set_bank_balances(&[
        Coin::new(12345, CONTRACT_DENOM),
        Coin::new(234, "ukrw"),
        Coin::new(345, "uusd"),
        Coin::new(69420,
"ibc/0471F1C4E7AFD3F07702BEF6DC365268D64570F7C1FDC98EA6098DD6DE59817B"),
    ]);

    execute(
        deps.as_mut(),
        mock_env_at_timestamp(35000),
        mock_info("worker", &[]),
        ExecuteMsg::Reconcile {},
    )
    .unwrap();
}
```