Audit Report

Hydro

v1.0

July 30, 2024

Table of Contents

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.

THIS AUDIT REPORT WAS PREPARED EXCLUSIVELY FOR AND IN THE INTEREST OF THE CLIENT AND SHALL NOT CONSTRUE ANY LEGAL RELATIONSHIP TOWARDS THIRD PARTIES. IN PARTICULAR, THE AUTHOR AND HIS EMPLOYER UNDERTAKE NO LIABILITY OR RESPONSIBILITY TOWARDS THIRD PARTIES AND PROVIDE NO WARRANTIES REGARDING THE FACTUAL ACCURACY OR COMPLETENESS OF THE AUDIT REPORT.

FOR THE AVOIDANCE OF DOUBT, NOTHING CONTAINED IN THIS AUDIT REPORT SHALL BE CONSTRUED TO IMPOSE ADDITIONAL OBLIGATIONS ON COMPANY, INCLUDING WITHOUT LIMITATION WARRANTIES OR LIABILITIES.

This audit has been performed by

Oak Security GmbH

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

Introduction

Purpose of This Report

Oak Security GmbH has been engaged by Simply VC Limited to perform a security audit of Hydro.

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/informalsystems/hydro
Commit	`0a7f6a957ff8e5c8e28689343e0ca5d0ba83c2d5`
Scope	All contracts were in scope.
Fixes verified at commit	`ede8b377eeadda15c810fbdf0af75a39b5cd4b1f` Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

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

Hydro is a protocol-owned liquidity mechanism where users can stake ATOM in exchange for voting powers. These voting powers enable users to vote on proposals that determine which decentralized exchanges should receive ATOM liquidity based on the number of votes received.

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-High	-
Code readability and clarity	Medium	-
Level of documentation	High	Documentation is available in the `README` file, the `docs` directory, the litepaper, and a forum post.
Test coverage	Medium	`cargo tarpaulin` reports a coverage of `74.75%`.

Summary of Findings

No	Description	Severity	Status
1	Voting and unlocking tokens may fail due to an out-of-gas error	Critical	Resolved
2	Unfinished development	Minor	Acknowledged
3	Missing `lock_epoch_length` and `round_length` validations	Minor	Acknowledged
4	`RoundProposals` query may fail due to running out of gas	Minor	Resolved
5	`ProposalTributes` query will not return legitimate tributes if the total tributes are larger than `DEFAULT_MAX_ENTRIES`	Minor	Resolved
6	Vulnerable dependency	Minor	Acknowledged
7	Token lockup grief	Informational	Acknowledged
8	Owner is unable to manage privileged roles	Informational	Acknowledged
9	Missing validation of duplicate entries	Informational	Resolved
10	The step function that scales the voting power can be smoothed or bypassed with a liquid lockup contract	Informational	Acknowledged
11	Misleading comment	Informational	Resolved
12	Hardcoded contract names may differ from cargo package names	Informational	Resolved
13	Lack of detailed event attributes decreases user experience	Informational	Acknowledged
14	Denom validation can be simplified	Informational	Resolved
15	Misleading parameter name	Informational	Resolved
16	Miscellaneous comments	Informational	Resolved

Detailed Findings

1. Voting and unlocking tokens may fail due to an out-of-gas error

Severity: Critical

The `unlock_tokens` and `vote` functions in `contracts/atom_wars/src/contract.rs:495-512` and `284-293`, respectively, iterate over all entries in the `LOCKS_MAP` state for the caller. This is problematic because if the user locks too many tokens in `contracts/atom_wars/src/contract.rs:159`, an out-of-gas error will occur when iterating the locks.

This may happen when users try to smoothen their power multiplier or if other contracts implement proxy voting.

Consequently, users cannot vote on proposals and unlock their tokens, causing a loss of funds.

Recommendation

We recommend implementing a maximum limit of entries that can be locked.

Status: Resolved

2. Unfinished development

Severity: Minor

In a few instances of the codebase, there seems to be unfinished development accompanied by TODO comments.

Firstly, there is no entry point to execute a proposal as it is commented out in `contracts/atom_wars/src/contract.rs:116-118`. This causes the `executed` field to become false, as seen in `contracts/atom_wars/src/state.rs:40`. There may be future logic that needs to be implemented, such as quorum participation validation that influences the proposal outcome.

Secondly, in `contracts/atom_wars/src/contract.rs:338-341`, the `validate_covenant_params` function does not perform any validation although the function name indicates that validation will be performed. This is problematic because if a user votes on a proposal with non-whitelisted `CovenantParams`, they will not be able to claim their tribute allocation in `contracts/tribute/src/contract.rs:180-187`. The `TopNProposals` query will exclude it from the top proposals, as seen in `contracts/atom_wars/src/contract.rs:787-796`.

Recommendation

We recommend completing development or removing unneeded code according to the business logic requirements.

Status: Acknowledged

The client states that they acknowledge this issue, but they deliberately leave these comments and fragments to guide the development of future iterations. The particular potentially problematic behavior that was pointed out (i.e., users not being able to claim tribute if they voted for non-whitelisted proposals) is intended. If the proposal is not whitelisted, it will not have any funds allocated to it, so the tribute should not be paid out.

There are legitimate reasons why users might want to vote for non-whitelisted proposals. For example, to demonstrate to whitelist administrators that a proposal has great support and should be whitelisted. Because users can always change their vote and see the whitelist, this is at worst suboptimal UX if users are not aware that a proposal they are voting for is not on the whitelist.

3. Missing `lock_epoch_length` and `round_length` validations

Severity: Minor

In `contracts/atom_wars/src/contract.rs:46`, `msg.lock_epoch_length` and `msg.round_length` are not validated to be within a reasonable range. Misconfiguring these values may cause users to have little time to vote on proposals, affecting the protocol's intended functionality.

Recommendation

We recommend validating the parameters to lie within a reasonable range. For example, the parameters could be validated to be longer than hours and smaller than years.

Status: Acknowledged

The client states that they do not believe it is appropriate to constrain the range by validating `msg.lock_epoch_length` and `msg.round_length`. The parameters will ultimately be chosen by governance or a multisig, and they want to give that entity the option to choose even very short or very long durations.

4. `RoundProposals` query may fail due to running out of gas

Severity: Minor

In `contracts/atom_wars/src/contract.rs:756-757`, the `query_round_tranche_proposals` function does not implement a pagination mechanism when it queries `PROPOSAL_MAP` for the current round/tranche combination.

Since anyone can create proposals, a malicious user can intentionally create many proposals, causing this query to fail due to an out-of-gas error and affecting third-party protocols that integrate with it.

Recommendation

We recommend implementing a pagination mechanism.

Status: Resolved

5. `ProposalTributes` query will not return legitimate tributes if the total tributes are larger than `DEFAULT_MAX_ENTRIES`

Severity: Minor

In `contracts/tribute/src/contract.rs:320`, the `query_proposal_tributes` function returns an array of tributes stored in the contract. However, if the total number of tributes is larger than `DEFAULT_MAX_ENTRIES`, any tributes added after that will not be returned to the caller.

This issue can be exploited by a malicious user who submits `DEFAULT_MAX_ENTRIES + 1` fake tributes to the contract to prevent external services from retrieving any legitimate tributes added after that.

A similar issue exists in `contracts/atom_wars/src/contract.rs:846`, affecting the `query_user_lockups` function.

Recommendation

We recommend adding pagination i.e. allowing the caller to pass a start and a count parameter to return `N` proposal tributes instead of `DEFAULT_MAX_ENTRIES`.

Status: Resolved

6. Vulnerable dependency

Severity: Minor

One of the dependencies of the audited code is affected by a publicly known security vulnerability:

- RUSTSEC-2024-0344

We classify this issue as minor as we did not find exploitable instances in the audited codebase. Further details on the outdated libraries can be found in Appendix A: Vulnerable dependency.

Recommendation

We recommend updating dependencies to the latest stable version.

Status: Acknowledged

The client states that since they need to be compatible with the versions of their dependencies supported by the chain they want to deploy on, they will investigate the option more closely before they launch the contract on mainnet.

7. Token lockup grief

Severity: Informational

The power scaling function is a step function. Consequently, marginally longer lockups drastically increase the raw voting power at the scaling steps. An above 6-month lockup grants a 4x increase, and a below 6-month lockup grants a 2x increase. Similarly, an above 3-month lockup grants a 2x increase, while a just below 3-month lockup leads only to a 1.5x increase.

This may result in strategic behavior and griefing incentives. Users with less raw voting power (grief voters) can time their lock so that they are locked only marginally (e.g., one round) longer than those with more voting power and can submit proposals that the ones with more raw voting power oppose. This forces the voters with more raw voting power to re-lock their tokens.

In anticipation of this, users might smoothen out their voting power curve. Alternatively, users could re-lock their tokens every round at the step, which yields maximal protection against this attack.

Both these mitigation strategies are costly as they require either frequently recurring calls or managing multiple locks.

We classify this issue as informational as the locked capital is not at risk, and the attack is costly for the grief voter.

Recommendation

We recommend implementing a smooth decaying function to reduce the incentive by increasing the capital needed for such an attack at every point of the power multiplier function.

Status: Acknowledged

The client states that they believe this is not an issue and that the protocol is working as intended. The grief voters still lock their tokens and accept longer locks in exchange for additional voting power.

8. Owner is unable to manage privileged roles

Severity: Informational

During the contract instantiation in `contracts/atom_wars/src/contract.rs:64-68`, a vector containing `WHITELIST_ADMINS` is stored. However, there is no entry point to manage that list to remove or add further admins. If the owner decides to remove an admin account due to loss or compromise or would like to add a new one, it will not be possible.

Recommendation

We recommend implementing logic to add and remove administrative accounts.

Status: Acknowledged

The client states that they assume the whitelist is managed via contract upgrades, so there is no extra functionality needed for this.

9. Missing validation of duplicate entries

Severity: Informational

During the contract instantiation in `contracts/atom_wars/src/contract.rs:63-69`, the deployer is able to specify the `WHITELIST_ADMINS` and `WHITELIST` vectors. Those vectors are not checked for duplicate entries.

Recommendation

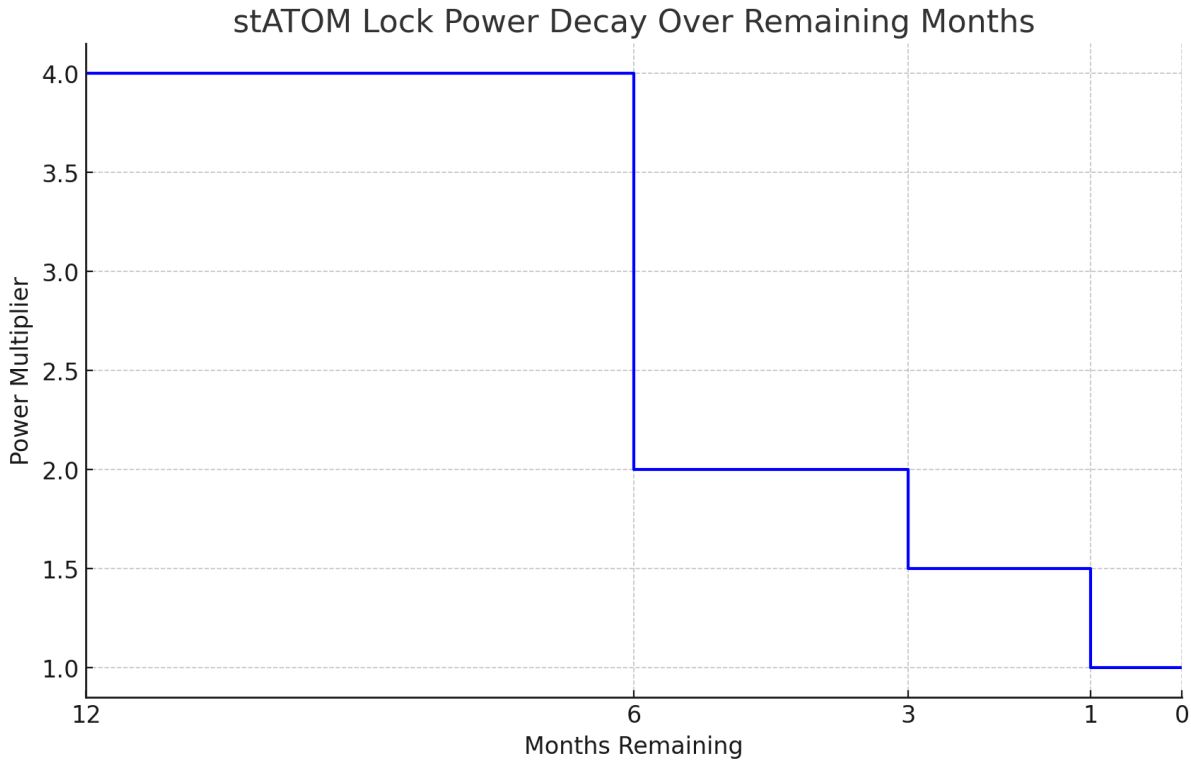
We recommend deduping the `WHITELIST_ADMINS` and `WHITELIST` vectors in the instantiation phase.

Status: Resolved

10. The step function that scales the voting power can be smoothed or bypassed with a liquid lockup contract

Severity: Informational

The protocol is designed with a step function that scales the raw voting power, which is displayed below.



The longer the user locks up their token, the more power they get. Apart from the opportunity costs of the locked funds, we did not identify economic risks (besides the smart contract risk) in locking the tokens. The step function leads to a sharp loss of voting power at the steps and no additional benefit for significantly longer locks than the step plus one round.

Users could relock their tokens every round close to the step or distribute their capital across multiple locks to smoothen out the power multiplier function.

Whether re-locking or smoothing is more desirable depends on the transaction costs, liquidity preference, and the perceived likelihood of grief attacks. Smoothing and managing multiple

locks will likely be too expensive for users with less funds and most attractive to high-value users, who might have a significant liquidity preference.

At a commercial scale, a liquid lockup contract could be designed to carry out proxy voting and offer liquidity of the locked-up token. Such a liquid-locked token could be used to create leverage by borrowing more lockable tokens in a pool to multiply the borrower's voting power even more. Such contracts could continuously re-lock the underlying token and smooth out the power multiplier function by offering a multiplier close to but below four and some short-term redeemable assets.

Additionally, an ecosystem or single users that leverage voting rights by borrowing and multiplying the voting power of borrowed funds could greatly amplify the threat of governance attacks. For example, a protocol offering 10x leverage and a 4x multiplier might amplify voting rights by a factor of 40.

Recommendation

We recommend using smaller multipliers to limit the effects of leverage and natively smoother scaling functions to make smoothing independent of the total amount of held capital.

Status: Acknowledged

The client states that they are aware of these limitations – liquid lockup contracts are not preventable in general. In fact, they think it is a benefit that other actors will be able to build their own infrastructure around the platform (e.g., to smooth out reward curves).

However, to lower the attack surface on the main contract, they prefer the current lock discrete lock function since it is cleaner to reason about and less prone to problems such as rounding errors.

11. Misleading comment

Severity: Informational

In `contracts/atom_wars/src/contract.rs:124`, the comment states that validation is performed against a `whitelist`. However, the protocol does not implement a whitelist but a single allowed denom, indicating the comment is inaccurate.

Recommendation

We recommend updating the comment to “Validation against allowed denom”.

Status: Resolved

12. Hardcoded contract names may differ from cargo package names

Severity: Informational

In `contracts/atom_wars/src/contract.rs:23` and `contracts/tribute/src/contract.rs:15`, the `CONTRACT_NAME` constant is set to a hardcoded string. However, if the cargo package name is modified, the `CONTRACT_NAME` is not reflect the updated cargo package name.

Recommendation

We recommend setting the `CONTRACT_NAME` constant using the `env!` macro, such as `CONTRACT_NAME: &str = env!("CARGO_PKG_NAME");`.

Status: Resolved

13. Lack of detailed event attributes decreases user experience

Severity: Informational

The following instances illustrate a lack of detailed events being emitted:

- In `contracts/atom_wars/src/contract.rs:176`, the `lock_tokens` function allows users to lock their tokens. However, the response events do not include any details about the lock. Consider adding more attributes to the response event of the `lock_tokens` function to include details about the `LockEntry` and the `sender`.
- In `contracts/atom_wars/src/contract.rs:382`, the `create_proposal` function allows users to create a new proposal. However, the response events do not include details about the created proposal. Consider adding more attributes to the response event of the `create_proposal` function to include details about the `Proposal` and the `sender`.
- In `contracts/tribute/src/contract.rs:37`, the `instantiate` function stores the tribute contract's initial state. However, the response events do not include details about the initial state. Consider adding more attributes to the response event of the `instantiate` function to include details about the `atom_wars_contract` and the `sender`.

Recommendation

We recommend applying the recommendations mentioned above.

Status: Acknowledged

The client states that their current plan is to coordinate the implementation of event attributes with their work on a frontend.

14. Denom validation can be simplified

Severity: Informational

In `contracts/atom_wars/src/contract.rs:135`, the `lock_tokens` function checks that the user sends the required denom and only one coin. However, this check is performed using a non-standard approach.

Recommendation

We recommend using the cw_utils::must_pay utility library instead.

Status: Resolved

15. Misleading parameter name

Severity: Informational

In `contracts/atom_wars/src/contract.rs:33`, the `instantiate` function's `_info` parameter is prefixed with an underscore, which means by convention that the parameter is not used in the function. However, the `_info` parameter is used in line `85` to set the sender value for one of the response attributes.

Recommendation

We recommend renaming the variable as `info`.

Status: Resolved

16. Miscellaneous comments

Severity: Informational

The following are suggestions to improve the overall code quality and readability.

- In `contracts/atom_wars/src/contract.rs:211`, consider moving the code so it is called before the `if` statement on line `205`. This enables the `old_lock_end` variable to be used in the `if` statement.
- In `contracts/atom_wars/src/contract.rs:324`, consider creating a new variable called `previous_round_id` between lines `318-319` to store the value of `current_round_id - 1`:

```
let previous_round_id = current_round_id - 1;
// use previous_round_id in loop
```

- In `contracts/atom_wars/src/contract.rs:375`, the `total_voted_power` variable is loaded from storage and checked if there is an error using `is_err`. Consider using a more efficient approach by utilizing the `has` function and removing the validation in line `378`:

```
if !TOTAL_VOTED_POWER.has(deps.storage, (round_id, tranche_id)) {
   // set default to zero
}
```

- In `contracts/atom_wars/src/contract.rs:385`, the `scale_lockup_power` function uses a `match` statement against the `lockup_time`. Consider replacing the `lockup_time` variable with an underscore in each `match` statement to improve readability:

```
match lockup_time {
      _ if lockup_time > lock_epoch_length * 6 => raw_power * two * two,
      // etc
}
```

- In `contracts/atom_wars/src/contract.rs:721`, the `query_user_voting_power` function allows clients to query a user's voting power. However, the function loads the constants from the `CONSTANTS` storage twice in lines `718` and `721`, which consumes unnecessary gas. Consider replacing line `721` with the following to decrease gas consumption:

```
let lock_epoch_length = constants.lock_epoch_length;
```

Recommendation

We recommend applying the recommendations mentioned above.

Status: Resolved

Appendix A: Vulnerable dependency

Please find below the raw output from the `cargo audit` command, showing the vulnerable dependency being used.

```
Crate:       curve25519-dalek
Version:   3.2.0
Title:       Timing variability in `curve25519-dalek`'s
`Scalar29::sub`/`Scalar52::sub`
Date:        2024-06-18
ID:          RUSTSEC-2024-0344
URL:         https://rustsec.org/advisories/RUSTSEC-2024-0344
Solution:  Upgrade to >=4.1.3
Dependency tree:
curve25519-dalek 3.2.0
└── ed25519-zebra 3.1.0
      └── cosmwasm-crypto 1.5.5
      └── cosmwasm-std 1.5.5
            ├── tribute 1.0.0
            ├── cw2 1.1.2
            │    ├── tribute 1.0.0
            │    └── atom-wars 1.0.0
            │      └── tribute 1.0.0
            ├── cw-storage-plus 1.2.0
            │    ├── tribute 1.0.0
            │    ├── cw2 1.1.2
            │    └── atom-wars 1.0.0
            └── atom-wars 1.0.0
```