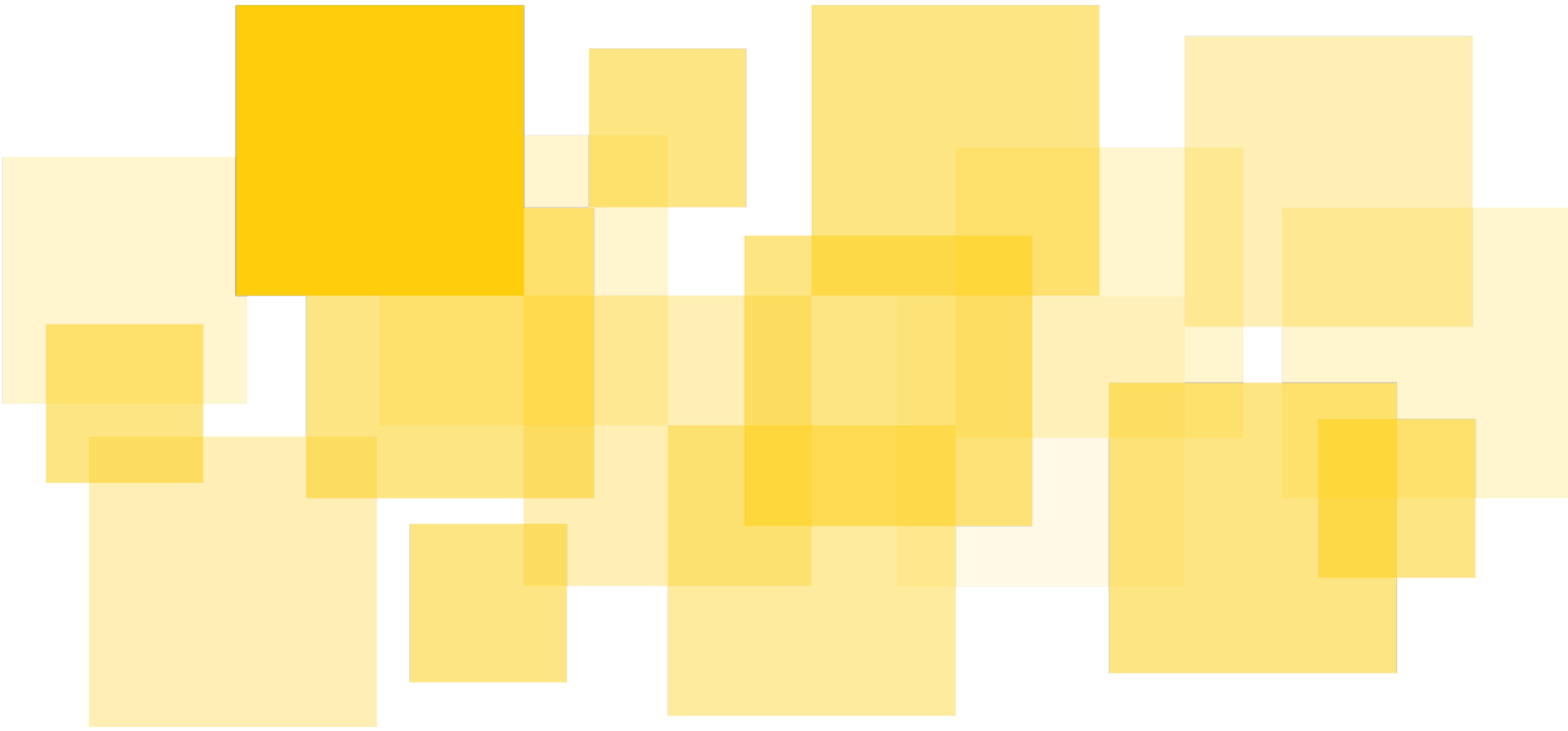
Security Audit Report

Hatom Protocol

September 29th, 2022



Prepared for Elrond Network by

runtime verification

Contents

Summary

Hatom protocol engaged Runtime Verification Inc to conduct a security audit of the smart contracts implementing their decentralized lending and borrowing market on the Elrond blockchain.

The objective was to review the contracts' business logic and implementation and identify any issues that could potentially cause the system to malfunction or be exploited.

The audit led to identifying 7 findings and 11 informative findings. We generally found the protocol to be thoughtfully engineered and collaborated very well with the Hatom team.

Scope

The scope of this review focuses on the following commits of the two repositories.

- Hatom-protocol - commit 6b1e8f6c42527aae71a7f113c3da5ff651aa0afc
- Hatom-tokenomics - commit f11dbb34bf3f725090867cc63896c7167f6c8ee7

Specifically, the audit was conducted on the artifacts in the following folder provided by the Hatom team.

- Hatom-protocol / common / admin / src
- Hatom-protocol / controller / src
- Hatom-protocol / governance / src
- Hatom-protocol / interest-rate-model / src
- Hatom-protocol / money-market / src
- Hatom-protocol / oracle / src
- Hatom-protocol / staking / src
- Hatom-tokenomics / src

Disclaimer

This report does not constitute legal or investment advice. You understand and agree that this report relates to new and emerging technologies and that there are significant risks inherent in using such technologies that cannot be completely protected against. While this report has been prepared based on data and information that has been provided or is otherwise publicly available, there are likely additional unknown risks which otherwise exist. This report is also not comprehensive in scope, excluding a number of components critical to the correct operation of this system. This report is for informational purposes only and is provided on an "as-is" basis and you acknowledge and agree that you are making use of this report and the information contained herein at your own risk. The preparers of this report make no representations or warranties of any kind, either express or implied, regarding the information in or the use of this report and shall not be liable to you or any third parties for any acts or omissions undertaken by you or any third parties based on the information contained herein.

Contract Description and invariants

Admin module

The admin module implements the functionality for the `admin` role of the smart contracts.

- The `set_admin` function should only be called within the `init` function.
- Changing admin address should go through the `set_pending_admin` and the `accept_admin` function.

Controller module

The controller module validates permitted user actions and disallows actions if they do not conform to certain risk parameters.

Governance

The `governance.rs` implements multiple `set` functions to update the parameters of the controller.

- The set functions can only be called by admin address.
- `grant_rewards` can only be called by admin address.

Guardian

The `guardian.rs` implements pause and unpause functions for mint, borrow and seize.

- The functions can only be called by admin or guardian address.

Market

The `market.rs` implements functions to provide and withdraw collateral.

- Users can only provide or withdraw collateral from the whitelisted money market.
- `update_supply_market_state` and `distribute_supplier_rewards` should be called when entering or exiting the market.

- `account_assets(A)` should always store the union of the borrowed assets and collateral assets of account A.

Policies

The `policies.rs` defines a list of policies to be checked before users take actions like borrow, repay, redeem, liquidate and seize.

- `mint` operation is allowed only if the status is `Active.`
- `redeem` operation is allowed if the risk profile is solvent after withdrawing the collateral.
- `borrow` operation is allowed if the status is `Active`, the borrow cap is not reached and the risk profile is solvent after the borrow.
- `repay` operation is allowed at any time.
- `liquidate` operation is allowed if the target money market is deprecated or the risk profile of the target borrower is solvent.
- `seize` operation can only be called from the money market and the status is `Active`.

Rewards

The `rewards.rs` implements functionality to distribute reward tokens based on the borrow and collateral amount.

- `update_borrow_market_state, distribute_borrower_rewards, update_supply_market_state, distribute_supplier_rewards` should be called when claiming rewards.
- `supply_state.index` is increasing over time if `supply_speed > 0`.
- `borrow_state.index` is increasing over time if `borrow_speed > 0`.
- `update_borrow_market_state, distribute_borrower_rewards, update_supply_market_state, distribute_supplier_rewards` should be called when setting rewards speed.
- The borrower rewards from the last claim timestamp to the current timestamp are distributed according to the proportion of the user's borrowed amount to the total borrowed amount.
- The supplier rewards from the last claim timestamp to the current timestamp are distributed according to the proportion of the user's supplied amount to the total supplied amount.
- `supply_state.index` scales up by multiplying a constant `10^36`.
- `borrow_state.index` scales up by multiplying a constant `10^36`.

Risk profile

The `risk_profile.rs` checks if an account is risky by iterating over its borrow and collateral.

- A user's profile is `Solvent` if `total_collateral >= total_borrow`. Otherwise, it is `RiskyOrInsolvent`.
- `close_factor` scales up by multiplying a constant `10^18`.
- `collateral_factor` scales up by multiplying a constant `10^18`.

Governance Module

The governance module handles proposing, voting and executing proposals.

- Proposal will be executed only when `total_upvotes - total_downvotes >= quorum` after the voting period ends.
- Users can only withdraw the staked HTM token after the voting period ends.

Interest Rate Module

The interest rate depends on the current utilization rate of a given market.

- `utilization_rate` scales up by multiplying a constant `10^18`.
- `0 <= utilization_rate.` In the extreme case, the `utilization_rate` can be greater than `10^18 (when borrowers borrow from the reserves).`
- `borrow_rate` scales up by multiplying a constant `10^18`.
- `0 < optimal_utilization < 10^18.`

Money Market Module

The money market module handles borrow, lend, repay and liquidate.

- `borrow_index` scales up by multiplying a constant `10^18`.
- `exchange_rate` scales up by multiplying a constant `10^18`.
- `borrow_index` is increasing over time.
- `exchange_rate` is increasing over time.
- `cash <= get_esdt_balance(self, token).`
- `Liquidation_incentive > 10^18.`

- `(liquidation_incentive / wad) * (1 - protocol_seize_share / wad) > 1`

Oracle

The oracle module facilitates obtaining prices of the assets involved measured in egld. A robust oracle module can protect the protocol from price manipulation. The oracle module fetches prices from 3 sources: `Maiar Reserves`, `Maiar SafePrice`, `Chainlink Price.` `Maiar Reserves` returns the instant ratio between token reserves and egld reserves in the pair pool. Therefore, `Maiar Reserves` is vulnerable to price manipulation attacks. `Maiar SafePrice` computes the `TWAP`(time-weighted average price) of a Maiar pair pool. `Chainlink Price` fetches the price from off-chain sources.

The oracle module comes with 4 modes:

- `Default`: the oracle module compares `Maiar SafePrice with Chainlink Price.` If two prices are within a tolerance range, `Chainlink Price` is recorded and returned. Otherwise, the price from the previous query is used and `guardian_price_event` is generated. The Hatom team will react to `guardian_price_event` and take off-chain operations. However, it is not within the scope of this audit. Moreover, the Hatom team will also monitor the chainlink service to make sure it reports the latest price.
- `Instantaneous:` return and record the `Maiar Reserves`.
- `Safe:`return and record the `Maiar SafePrice`.
- `Chainlink:`return and record the `Chainlink Price`.

Invariants:

- `Price > 0`

Staking Module

The staking module distributes staking rewards to the Hatom token stakers.

- `stake_state.index` is increasing over time.

Findings

A01: Wrong money market is updated in the `seize_allowed` function

[Severity: Medium | Difficulty: Low | Category: Functional Correctness]

During the seizure operation, the collateral amount of the borrower, the collateral amount of the liquidator and the total collateral amount will change in the collateral money market. Therefore, the supply market state should be updated for the collateral money market instead of the borrow money market. This issue combined with the issue A02 makes it hard to find during testing.

Recommendation

Pass parameter `collateral_amm` to `update_supply_market_state` and `distribute_supplier_rewards`.

Status

The issue was fixed in PR #9.

A02: `seize_allowed` function is called with wrong parameter order

[Severity: Medium | Difficulty: Low | Category: Functional Correctness]

When calling the `seize_allowed` function, the first parameter is `borrow_mma` and the second parameter is `collateral_mma`. In the context of the `seize_internal` function, the `borrow_mma` should be `from_money_market` and the `collateral_mma` should be `this_money_market`.

Recommendation

Pass the parameters to the `seize_allowed` function in the correct order.

Status

The issue was fixed in the PR #9.

A03: Staking contract may not have sufficient Hatom token to vote

[Severity: Medium | Difficulty: Low | Category: Functional Correctness]

Users can stake both Hatom token(HTM) and Locked-Hatom(LHTM) token to the staking contract. When a user uses staked tokens to vote for proposals, the contract checks against the total staked tokens (HTM+LHTM). However, the LHTM token is not accepted by the governance contract.

Recommendation

The staked LHTM token can not be used for voting.

Status

The issue was fixed in the PR #13.

A04: `update_stake_market_state` function does not use the latest stake rewards

[Severity: Medium | Difficulty: Low | Category: Functional Correctness]

The `update_stake_market_state` function calls `get_total_rewards(&money_market)` which only reads the `staking_rewards` field in the money market. Since `update_stake_market_state` doesn't directly or indirectly call the `accrue_interest` function from the money market, it will not use the latest staking rewards to update the `stake_state.index`.

Recommendation

`get_total_rewards` function should call the `accure_interest` function in order to return the latest stake rewards.

Status

The issue was fixed in the PR #11.

A05: `update_stake_market_state` function may miss some staking rewards to distribute

[Severity: Medium | Difficulty: Low | Category: Functional Correctness]

`update_stake_market_state` function calls `get_total_rewards` which returns the current stake rewards stored in the money market contract. The value will become 0 after users claim the rewards. The function misuses this value and it can miss some staking rewards to distribute.

Consider the following scenario:

At `T1`, `claim_rewards_markets_stakers` function is called. Inside the function, it calls `update_stake_market_state` to update the `stake_state.rewards` to `R1`.

At `T2`, `claim_rewards_markets_stakers` is called again. Between `T1` and `T2`, some staking rewards `R2` are accumulated in the money_market contract. The correct implementation should distribute `R2` to the stakers.

However, in the current implementation,

- if `R2 < R1`, then no reward is distributed
- if `R2 > R1`, then only R2-R1 is distributed

Recommendation

The money market contract can track the historical staking rewards, which is a monotonic nondecreasing function. The update_stake_market_state can just use that variable for computing the amount of staking rewards that should be distributed.

Status

The issue was fixed in PR #12.

A06: Staking rewards is not ensured when borrowing money or reducing reserves from a money market

[Severity: Medium | Difficulty: Low | Category: Functional Correctness]

`cash` field in the money market include both the money that is available to be borrowed and the protocol reservers (protocol income + staking rewards). When borrowing money or reducing reserves from a money market, it only checks that `cash` is greater than or equal to the desired amount

Recommendation

Check that `cash - staking_rewards >= desired_amount`

Status

The issue was fixed in PR #10.

A07: Proposals can be published or executed multiple times

[Severity: Medium | Difficulty: Low | Category: Functional Correctness]

The `publish_proposal` and `execute` function do not check if the proposal has been published or executed.

Recommendation

Check if the proposal has been published in the `publish_proposal` function and check if the proposal has been executed in the `execute` function.

Status

The issue was fixed in PR #18.

A08: Protocol will get stuck when `borrow_rate` exceeds `max_borrow_rate`

[Severity: Medium | Difficulty: Low | Category: Functional Correctness]

In the extreme case where `borrow_rate > max_borrow_rate`, the `accrue_interest` function will fail and as a result, it will make the protocol stuck in the current state.

Recommendation

Emit an event when `borrow_rate > max_borrow_rate` instead of revert the transaction. The off-chain bot will take actions to the event.

Status

The issue was fixed in PR #24.

A09: The `get_updated_account_accrued_rewards` view function does not reflect the latest rewards

[Severity: Medium | Difficulty: Low | Category: Functional Correctness]

The `get_updated_account_accrued_rewards` calculates the user's accrued rewards until the last time when `supply_state/borrow_state` is updated.

Recommendation

Use the `supply_speed/borrow_speed` to calculate the rewards that need to be distributed from now to the last time when the state is updated. After that, use the new state to calculate the rewards.

Status

The issue was fixed in PR #25,

A10: The staked Hatom tokens are not locked when voting proposals

[Severity: Medium | Difficulty: Low | Category: Functional Correctness]

When users vote proposals, the hatom tokens will be locked until the voting period ends. However, when a user votes through the staking contract, she can directly unstake to get all the hatom tokens back.

Recommendation

In the `unstake` function, exclude the amount which is used for voting when calculating the amount that is available to unstake.

Status

The issue was fixed in PR #30.

A11: `approve_address_change` should update `users_in_group` to replace old address with new address

[Severity: Low | Difficulty: Low | Category: Functional Correctness]

When replacing the old address with a new address, the `approve_address_change` function should also update `users_in_group` in order to make the data consistent.

Status

The issue was fixed in PR #1.

A12: Owner of the contract can change the admin by upgrading the contract

[Severity: Low | Difficulty: Low | Category: Security]

The init function will be called when upgrading the contract. The admin can be reset in the init function.

Recommendation

The init function should check if the admin address has already been set.

Status

The issue was fixed in PR #3.

Informative Findings

B01: Voters can not vote for two proposals at the same time

In the current design of the governance module, users need to lock Hatom token in the governance contract for the voting period. If there are two proposals at the same time, users may not have tokens to vote for the second proposal.

Status

The Hatom team followed the same approach implemented for the Maiar Governance, which they believe is a really good first implementation. The Hatom team has also made several improvements, such as enabling multiple actions in a proposal. In the future, the team might tackle this feature.

B02: Caller of the `get_money_market_identifiers` should check if the returned `token_id` is empty

When the get_money_market_indentifiers function is called, it is possible that the underlying money market hasn't issued a token yet. In this case, get_money_market_indentifiers will return an empty identifier.

Status

The issue was fixed in the PR #1.

B03: Optimize `tokens_to_seize` function

The `tokens_to_seize` function can be optimized as:

```
let num = &li * &borrow_price;
let den = &collateral_price * &fx / &wad;
let ratio = &num / &den;
```

Status

The issue was fixed in the PR #4.

B04: `[#view(...)]` annotation is misused

There are many places in the code where functions are annotated with #view but modify the contract state.

Status

The issue was fixed in the PR #5.

B05: Redundant check in the `try_change_governance_token_id` function

In the `try_change_governance_token_id` function,

```
self.governance_token_id().set(&token_id);
let new_token_id = self.governance_token_id().get();
require!(new_token_id == token_id, "tokens dont match");
```

The `new_token_id == token_id` check is redundant.

Status

The issue was fixed in the PR #6.

B06: `new_model_parameters_event` is not used

The `new_model_parameters_event` is defined here, but it is not used anywhere else in the project.

Status

The issue was fixed in the PR #7.

B07: The governance contract needs to support multi ESDT transfer and execute endpoint

The governance contract needs the capability to execute the action of multi ESDT token transfer and execute.

Status

For now, at this PR #8 , the team added an action for a governance proposal that sends a unique esdt token, as the team is interested in sending individual esdt tokens for the moment.

B08: Add `tokens > 0` check when exiting the market

Status

The issue was fixed in the PR #14.

B09: Avoid unnecessary scaling in `underlying_amount_to_tokens` function

Status

The issue was fixed in the PR #15.

B10: Avoid unnecessary scaling in `get_account_borrow_amount` function

Status

The issue was fixed in the PR #16.

B11: Users can not withdraw a collateral if she borrows the same asset

In the `exit_market` function, there is a check that prevents users from withdrawing the collateral if they have an outstanding borrow in the same asset to withdraw.

Status

The issue was fixed in the PR #17.

Other

In addition to the fixes above, we also reviewed the following PRs which are mainly refactors of the contract.

- #19
- #20
- #21
- #22
- #23
- #27

- #28
- #29
- #31
- #32