Security Audit Report

Hatom controller (3)
MultiversX smart contract

by ARDA

on January 17, 2025

Table of Content

Disclaimer

The report makes no statements or warranties, either expressed or implied, regarding the security of the code, the information herein or its usage. It also cannot be considered as a sufficient assessment regarding the utility, safety and bugfree status of the code, or any other statements.

This report does not constitute legal or investment advice. It is for informational purposes only and is provided on an "as-is" basis. You acknowledge that any use of this report and the information contained herein is at your own risk. The authors of this report 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.

Terminology

Code: The code with which users interact.

Inherent risk: A risk for users that comes from a behavior inherent to the code's design.

Inherent risks only represent the risks inherent to the code's design, which are a subset of all the possible risks. **No inherent risk doesn’t mean no risk.** It only means that no risk inherent to the code's design has been identified. Other kind of risks could still be present. For example, the issues not fixed incur risks for the users, or the upgradability of the code might also incur risks for the users.

Issue: A behavior unexpected by the users or by the project, or a practice that increases the chances of unexpected behaviors to appear.

Critical issue: An issue intolerable for the users or the project, that must be addressed.

Major issue: An issue undesirable for the users or the project, that we strongly recommend to address.

Medium issue: An issue uncomfortable for the users or the project, that we recommend to address.

Minor issue: An issue imperceptible for the users or the project, that we advise to address for the overall project security.

Objective

Our objective is to share everything we have found that would help assessing and improving the safety of the code:

1. The **inherent risks** of the code, labelled R1, R2, etc.
2. The **issues** in the **code**, labelled C1, C2, etc.
3. The **issues** in the **testing** of the code, labelled T1, T2, etc.
4. The **issues** in the **other** parts related to the code, labelled O1, O2, etc.
5. The **recommendations** to address each issue.

Audit Summary

Initial scope

- **Repository:** https://github.com/HatomProtocol/hatom-protocol
- **Commit:** `5a7edf3a9c41eb9bb0ea98c1cd207fcfadfc9416`
- **MultiversX smart contract path:** ./controller/

Final scope

- **Repository:** https://github.com/HatomProtocol/hatom-protocol
- **Commit:** `a4b070c0fa7f4eec2d6f3a825862ac29a0d7e960`
- **MultiversX smart contract path:** ./controller/

3 inherent risks in the final scope

1 issue in the final scope

9 issues reported in the initial scope and 1 remaining in the final scope:

Severity	Reported			Remaining		
	Code	Test	Other	Code	Test	Other
Critical	1	0	0	0	0	0
Major	3	0	0	0	0	0
Medium	4	0	0	1	0	0
Minor	1	0	0	0	0	0

Inherent Risks

R1: The solvency of a user might be incorrectly assessed, possibly leading to bad debt or to the liquidations of solvent users.

This is because the solvency of a user depends on the value of his collateral relative to the value of his debt, and the prices of these tokens are obtained from Hatom Oracle, thus there is a risk as for any oracle that incorrect prices are returned. Consequently:

- Insolvent users might be deemed solvent: This would prevent the liquidations of these users, and would also allow them to borrow assets or withdraw collateral, possibly creating bad debt and preventing lenders from withdrawing their funds.
- Solvent users might be deemed insolvent: This could result in unexpected liquidations, possibly making borrowers lose funds.

R2: Lenders have no guarantee that liquidations of insolvent borrowers will be timely performed.

This is because liquidations must be triggered by external accounts, therefore it is possible that at a time when some users are insolvent, there are no sufficiently active liquidators to perform liquidations or that prices fail to be obtained from the Oracle. This could in turn create bad debt and prevent lenders in the affected money markets from fully withdrawing their funds.

R3: Users may not be able to claim rewards as HTM if they claim too late.

This is because the contract has only a limited amount of rewards that can be converted to HTM.

Example: Let's say that if Alice claims now, she would be able to claim rewards as HTM. However, if she rather decides to claim one week later, it is possible that she may not be able to claim rewards as HTM anymore, for instance in the following cases:

- Other users have claimed rewards as HTM during the week, and there are not enough remaining rewards that can be converted to HTM for Alice.
- No other users claimed during the week, but Alice's rewards have increased and may have now exceeded the contract's amount of rewards that can be converted to HTM.

Code Issues & Recommendations

Since the code is not open-source, only the remaining issues are published.

C5: Liquidations and collateral withdrawals might exceed limits of gas and of calls to built-in functions

Severity: Medium **Status:** Won't fix

Description

Current behavior: Actions which change the user's collateral, e.g. liquidations and collateral withdrawals, have significant gas cost and call many built-in functions. Indeed, these actions perform multiple computations, storage updates, and calls to several external smart contracts. In particular, at the time of this audit, these actions call a new endpoint `on_market_change` of the USH money market which performs heavy logic as well.

However, if the gas cost of these actions is too high, e.g. superior to the gas half-limit in a mini-block (300M), or if the number of built-in calls is too high, e.g. the number of storage reads exceeds 1500, then the transactions could take too long to be executed or even fail, possibly resulting in losses of funds.

Moreover, past simulations have shown that liquidations and collateral withdrawals already cost significant amounts of gas. Therefore, changes increasing the gas costs of these actions are highly sensitive.

Expected behavior: The gas cost of actions involving collateral changes should stay below 300M gas, in order to give transactions enough chances to be executed, even in a context where the shard is saturated and at most 600M gas can be consumed by all transactions included in a block. In addition, these actions should not exceed the limits of built-in function calls, to make sure that transactions can be executed.

Worst consequence: If the gas cost issue exists, an attacker could exploit it by borrowing a huge amount and becoming impossible to liquidate, as any liquidation transaction would run out of gas.

Recommendation

At a high-level, we recommend running a system test of the worst case scenario on devnet in order to check that gas costs are reasonable, i.e. less than the gas half-limit in a mini-block (300M), and that the number of built-in calls is not too big, i.e. it does not make the transaction fail. Then, the devnet transactions can be shared with the auditor. Subsequently, if we observe that gas costs or the number of built-in calls are too big, then we propose approaches to reduce these amounts, otherwise it is reasonable to consider that there is no issue.

More precisely, here are two test scenarios. The 1st one assumes that the market observers in the Controller are the USH money market and the Booster v2:

- Add the maximal number of discounts on interest in the Discount Rate Model. Note that from the issue *"List of discounts can be too big and users can't be liquidated or withdraw their collateral"* reported in the audit report of the Discount Rate Model, this amount should be bounded.
- Create two user accounts, Alice and Bob, who are USH borrowers, with collateral across the maximum number of money markets: `MAX_MARKETS_PER_ACCOUNT = 8`.
- Create the maximum number of active rewards batches in the Controller for these money markets: `MAX_REWARDS_BATCHES = 3`.
- Create the maximum number of active rewards batches in the Booster for these money markets: `MAX_REWARDS_BATCHES = 2`.
- Alice stakes in the Booster the maximum number of different j-tokens: `MAX_ACCOUNT_STAKE_TOKENS = 3`.
- Bob liquidates Alice. Bob also has non-zero collateral in the same money market where the liquidation occurs.
- We check that the overall gas cost of the transaction is at most around `300M` gas.

The 2nd scenario is the same as the 1st one except that the Booster v1 is used instead of the Booster v2, and Alice and Bob start with a balanced portfolio in the Booster.

Subsequently, in case one test witnesses that the transaction fails or costs significantly more than 300M gas, in order to resolve the issue, we can reduce

the gas costs and number of built-in calls of the endpoints `on_market_change`, both in the Booster and in the USH money market, as well as in the endpoints of the Controller, as described below.

1) In the Controller, we can introduce a smaller limit on the number of money markets a user is allowed to enter, specifically for USH borrowers, e.g. a limit of 4 money markets, including the USH money market.

2) In the Booster, we can follow the suggestions in the issue *"“unstake” and “on_market_change” might consume too much gas"* of the Booster v2 audit report.

3) In the USH money market, we can implement some of the following changes:

- Not re-computing the discount on the interest twice for the liquidator, as done currently: once after the debt is repaid, and once after the collateral is seized. It would be sufficient to compute the discount only once in the end.
- Not updating the discount of the liquidator in a liquidation transaction. The liquidator would update his discount later by himself if he wants, as it will be in his favor.
- Allowing only 1 collateral for computing the discount, and let the user choose the collateral he wants to use. With this way of doing, in `on_market_change`, there would be no need to load the Discount Rate Model smart contract and execute complex logic to compute the user's discount, as it would be enough to read the storage of the discount data associated to that collateral and use it directly.

Resolution notes

Multiple optimizations have been implemented, allowing to reduce the gas costs of the worst case scenario to 472M (link to devnet transaction). Although this is a significant improvement, this gas cost remains significantly superior to 300M, i.e. half the limit of a mini-block, hence it might be difficult to include such liquidation transactions when the blockchain is congested.