// HALBORN

Hatom Protocol

MultiverseX Smart Contract Security Assessment

Prepared by: **Halborn**

Date of Engagement: **March 7th, 2023 - April 10th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY 4
CONTACTS 5
1 EXECUTIVE OVERVIEW 6
1.1 INTRODUCTION 7
1.2 ASSESSMENT SUMMARY 8
1.3 TEST APPROACH & METHODOLOGY 9
RISK METHODOLOGY 10
1.4 SCOPE 12
2 ASSESSMENT SUMMARY & FINDINGS OVERVIEW 14
3 FINDINGS & TECH DETAILS 15
3.1 (HAL-01) HARDCODED USDC DOLLAR PEG IN CHAINLINK PRICE CALCULATION - MEDIUM(6.3) 17
Description 17
Code Location 17
BVSS 18
Recommendation 18
Remediation Plan 19
3.2 (HAL-02) REWARDS PERIOD SHOULD BE GREATER THAN ZERO - LOW(3.1) 20
Description 20
Code Location 20
BVSS 21
Recommendation 21
Remediation Plan 21
3.3 (HAL-03) SINGLE STEP OWNERSHIP TRANSFER PROCESS - LOW(3.3) 22

Description 22
Code Location 22
BVSS 22
Recommendation 22
Remediation Plan 23
3.4 (HAL-04) ANCHOR TOLERANCE DOES NOT HAVE A THRESHOLD IN THE ORACLE CONSTRUCTOR - INFORMATIONAL(0.0) 24
Description 24
Code Location 24
BVSS 25
Recommendation 25
Remediation Plan 25
3.5 (HAL-05) REDUNDANT CHECK FOR UNDERLYING AMOUNT GREATER THAN ZERO - INFORMATIONAL(0.0) 26
Description 26
Code Location 26
BVSS 26
Recommendation 26
Remediation Plan 27
3.6 (HAL-06) MISSING INTEREST RATE MODEL VALIDATION - INFORMATIONAL(0.0) 28
Description 28
Code Location 28
BVSS 28
Recommendation 29
Remediation Plan 29

3.7 (HAL-07) SUPPORTED PAIRS CANNOT BE MADE UNSUPPORTED - INFORMATIONAL(0.0) 30
Description 30
BVSS 30
Recommendation 30
Remediation Plan 30
4 MANUAL TESTING 31
4.1 INTEREST RATE MODEL 32
4.2 MONEY MARKET 32
4.3 ORACLE 32
4.4 CONTROLLER 32
4.5 ADMIN 33
4.6 GOVERNANCE 33
5 AUTOMATED TESTING 33
5.1 AUTOMATED ANALYSIS 35
Description 35

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE
0.1	Document Creation	03/24/2023
0.2	Document Updates	03/24/2023
0.3	Document Updates	03/31/2023
0.4	Document Updates	04/07/2023
0.5	Draft Review	04/11/2023
0.6	Draft Review	04/11/2023
0.7	Draft Review	04/12/2023
1.0	Remediation Plan	04/21/2023
1.1	Remediation Plan	06/13/2023
1.2	Remediation Plan Review	06/13/2023
1.3	Remediation Plan Review	06/13/2023
2.0	Document Updates	06/26/2023
2.1	Document Updates	06/28/2023
2.2	Document Updates Review	06/30/2023
2.3	Document Updates Review	06/30/2023
2.4	Document Updates Review	06/30/2023

3.0	Remediation Plan	07/09/2023
3.1	Remediation Plan Review	07/10/2023
3.2	Remediation Plan Review	07/10/2023
3.3	Remediation Plan Update	10/26/2023
3.4	Remediation Plan Update Review	10/26/2023

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Hatom Protocol is a decentralized lending protocol developed on the MultiversX (Elrond) blockchain, which aims to provide a comprehensive money market ecosystem to users. The platform enables users to lend, borrow, and stake various cryptocurrencies at interest rates determined by supply and demand dynamics in the market. The protocol offers depositors the opportunity to earn passive income by providing liquidity, while borrowers can access loans by supplying collateral in an over-collateralized fashion, which reduces the risk of default.

Hatom Protocol ’s ecosystem is designed to support eGold (EGLD), the native token of MultiverseX, along with other major cryptocurrencies, facilitating seamless integration and interaction between the two platforms. This compatibility with Multiversx’s native token ensures that users can access a wide range of digital assets within the lending protocol.

The platform may also implement a governance model to maintain decentralization and encourage community involvement. This governance system would allow token holders to participate in decision-making processes, such as proposing and voting on protocol updates or modifications.

In summary, Hatom Protocol is a decentralized lending protocol on the Multiversx (Elrond) blockchain that offers users the ability to lend, borrow, and stake cryptocurrencies at market-driven interest rates. The platform seeks to provide a reliable and efficient DeFi solution by leveraging the advanced features of the Multiversx (Elrond) network, while promoting financial inclusion and accessibility in the digital asset space.

Initially Hatom Protocol engaged Halborn to conduct a security assessment on their smart contracts beginning on March 7th, 2023 and ending on April 10th, 2023 . Before launching the protocol, some changes, and functionalities were added to the code and Halborn conducted a review on the new code beginning on 06/26/2013 and ending on 06/30/2023.

1.2 ASSESSMENT SUMMARY

The team at Halborn was provided four weeks for the engagement and assigned two full-time security engineers to verify the security of the smart contracts. The security engineers are blockchain and smart-contract security experts with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this assessment to achieve the following:

- Ensure that smart contract functions operate as intended: Verify that each function performs its intended purpose and adheres to the specified requirements.
- Identify potential security issues with the smart contracts: Detect vulnerabilities, bugs, and weaknesses that could be exploited by malicious actors or result in unintended consequences.
- Verify compliance with best practices and coding standards: Check that the smart contract code follows established best practices, such as clear documentation, proper error handling, and efficient gas usage.
- Evaluate the overall design and architecture: Assess the smart contract's design and structure to ensure it is robust, modular, and maintainable.
- Validate access controls and permissions: Ensure that access controls are correctly implemented and that only authorized parties can perform sensitive operations.
- Confirm proper handling of user funds and assets: Verify that the smart contract securely manages user funds and assets without risk of loss or unauthorized access.
- Test for potential race conditions and reentrancy attacks: Identify scenarios where concurrent transactions or external calls could lead to unexpected behavior or security vulnerabilities.

- Examine the smart contract's upgradeability and extensibility: Ensure that the contract can be upgraded or extended in a secure and controlled manner, if necessary.
- Review the contract's interactions with other contracts and external systems: Analyze how the smart contract interacts with other components in the ecosystem, including other smart contracts and external services, to identify potential issues or points of failure.
- Validate the smart contract's performance and resource usage: Analyze the contract's gas consumption and resource usage to ensure it operates efficiently and does not impose unnecessary costs on users or the network.

In summary, Halborn identified some improvements to reduce the likelihood and impact of multiple issues which were successfully addressed by the Hatom team. The main ones were the following:

(HAL-01) HARDCODED USDC DOLLAR PEG IN CHAINLINK PRICE CALCULATION
The get_chainlink_price_in_egld function assumes that the USDC stablecoin is always pegged 1:1to USD, which may not always be true. This assumption can lead to inaccurate price calculations, potentially impacting users' collateral values, borrowing capacity, and liquidations.

(HAL-02) REWARDS PERIOD SHOULD BE GREATER THAN ZERO
In the function update_rewards_batch_speed, there is a possibility that the calculated rewards batch period dt is zero. This may lead to unexpected behavior, such as the rewards batch being stuck in the current state and preventing further rewards distribution.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of the manual view of the code and automated security testing to balance efficiency, timeliness, practicality, and accuracy regarding the scope of the smart contract assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation, automated testing techniques help enhance the coverage

of smart contracts. They can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the assessment:

- Research into architecture, purpose, and use of the platform.
- Manual code read and walk through.
- Manual Assessment of use and safety for the critical Rust variables and functions in scope to identify any arithmetic related vulnerability classes.
- Race condition tests.
- Cross contract call controls.
- Architecture related logical controls.
- Fuzz testing. (cargo fuzz)
- Checking the unsafe code usage. (cargo-geiger)
- Scanning of Rust files for vulnerabilities.(cargo audit)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

Code repositories:
Note: Before the launch of the Hatom protocol, several changes were made, and new functionalities were added to the codebase. In response, Halborn conducted a review of the updated code. Additionally, the governance contract was moved to a different repository and the changes to it were verified too.

Halborn started the assessment with 44bd3, then checked the first assessment remediation's in eaaf9.

Halborn conducted a second follow-up assessment on Hatom protocol's new updates, which was scoped as: 4fd4e and 9d958.

1. Hatom Protocol
    - Repository: HatomProtocol
    - Commit ID 1: 44bd3
        - Smart Contracts in scope:
            1. Interest Rate Model
            2. Money Market
            3. Oracle
            4. Controller
            5. Admin
            6. Governance

            Out-of-scope: Staking, Safety Module
    - Commit ID 2: eaaf9
        - Smart Contracts in scope:
            1. Interest Rate Model
            2. Money Market
            3. Oracle
            4. Controller
            5. Admin

            Out-of-scope: Staking, Safety Module
    - Commit ID 3: 4fd4e

    - Smart Contracts in scope:
        1. Interest Rate Model
        2. Money Market
        3. Oracle
        4. Controller
        5. Admin

        Out-of-scope: Staking, Safety Module
    - Commit ID 4: 8c5cd
        - Smart Contracts in scope:
            1. Interest Rate Model
            2. Money Market
            3. Oracle
            4. Controller
            5. Admin

            Out-of-scope: Staking, Safety Module

2. Hatom Governance

- Repository: hatom-governance
- Commit ID 1: 9d958
- Commit ID 2: 7eab0

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	2	4

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) HARDCODED USDC DOLLAR PEG IN CHAINLINK PRICE CALCULATION	Medium (6.3)	SOLVED - 06/28/2023
(HAL-02) REWARDS PERIOD SHOULD BE GREATER THAN ZERO	Low (3.1)	SOLVED - 06/29/2023
(HAL-03) SINGLE STEP OWNERSHIP TRANSFER PROCESS	Low (3.3)	SOLVED - 05/26/2023
(HAL-04) ANCHOR TOLERANCE DOES NOT HAVE A THRESHOLD IN THE ORACLE CONSTRUCTOR	Informational (0.0)	SOLVED - 04/15/2023
(HAL-05) REDUNDANT CHECK FOR UNDERLYING AMOUNT GREATER THAN ZERO	Informational (0.0)	SOLVED - 04/15/2023
(HAL-06) MISSING INTEREST RATE MODEL VALIDATION	Informational (0.0)	SOLVED - 03/29/2023
(HAL-07) SUPPORTED PAIRS CANNOT BE MADE UNSUPPORTED	Informational (0.0)	SOLVED - 06/08/2023

FINDINGS & TECH DETAILS

3.1 (HAL-01) HARDCODED USDC DOLLAR PEG IN CHAINLINK PRICE CALCULATION - MEDIUM (6.3)

Description:

The get_chainlink_price_in_egld function assumes that the USDC stablecoin is always pegged to USD, which may not always be true. This assumption can lead to inaccurate price calculations, potentially impacting users' collateral values, borrowing capacity, and liquidations; particularly during periods of high market volatility or external factors causing deviations from the 1:1 peg. The inaccurate price calculations can have consequences such as:

- Unfair treatment of users
- Incorrect interest rates
- Unjust liquidations

Code Location:

Down below is the code snippet from the get_chainlink_price_in_egld function:

Listing 1: oracle/src/prices.rs (Line 130)

```
125    #[view(getChainlinkPriceInEgld)]
126     fn get_chainlink_price_in_egld(&self, token_id: &
 ↳ TokenIdentifier) -> BigUint {
127         let optional_pair = self.get_chainlink_pair(token_id);
128         match optional_pair {
129             None => sc_panic!(ERROR_UNSUPPORTED_TOKEN),
130             Some(pair) => {
131                 let wad = BigUint::from(WAD);
132                 let (_, _, _, _, egld_in_usd, decimals0) = self.
 ↳ get_chainlink_latest_price_feed(&ManagedBuffer::from(
 ↳ CHAINLINK_EGLD_SYMBOL), &ManagedBuffer::from(CHAINLINK_USD_SYMBOL)
 ↳ );
133
```

```
134                 // chainlink is not pushing the USDC price, assume
 ↳  USDC is always pegged to USD
135                 let (token_in_usd, decimals1) = if pair.
 ↳ chainlink_symbol.to_boxed_bytes().as_slice() == b"USDC" {
136                     (wad.clone(), 18u8)
137                 } else {
138                     let (_, _, _, _, token_in_usd, decimals1) =
 ↳ self.get_chainlink_latest_price_feed(&pair.chainlink_symbol, &
 ↳ ManagedBuffer::from(CHAINLINK_USD_SYMBOL));
139                     (token_in_usd, decimals1)
140                 };
141                 ...snipped...
142                 self.chainlink_price_fetched_event(token_id, &
 ↳ price);
143                 price
144             },
145         }
146     }
```

BVSS:

AO:A/AC:L/AX:M/C:N/I:N/A:N/D:H/Y:H/R:N/S:U (6.3)

Recommendation:

To mitigate the potential impact of peg deviation, consider fetching the USDC price directly from a trusted oracle, rather than assuming a constant 1:1 peg. This ensures that the lending platform utilizes accurate price data, accounting for any deviation from the peg in its calculations. Implementing monitoring and alerting mechanisms to detect substantial deviations from the expected peg can also help take corrective actions in a timely manner. Lastly, incorporating additional checks and safety measures in the lending platform's logic, such as collateral buffer requirements, can provide further protection against the risks posed by peg deviation.

Remediation Plan:

SOLVED: The issue was solved in commit 58c1c by removing the hardcoded price and requesting the price from Chainlink instead.

3.2 (HAL-02) REWARDS PERIOD SHOULD BE GREATER THAN ZERO - LOW (3.1)

Description:

In the function update_rewards_batch_speed, there is a possibility that the calculated rewards batch period dt could be zero. This may lead to unexpected behavior, such as the rewards batch being stuck in the current state and preventing further rewards distribution.

The problem arises from the fact that there is no explicit check for whether dt is zero, and the code proceeds to update the rewards_batch.end_time with the potentially zero value.

Consider the following scenario:

A rewards batch has an initial speed of 100, old_dt of 100, and new_speed of 10,000.
The calculation for new_dt would then be:

$$(100 * 100 * wad)/(10,000 * wad) = 1$$

In this case, the value of dt is 1, which is greater than zero and proceeds normally.
However, if we change the initial speed to 1 and the new_speed to 10,000, the calculation for new_dt would result in:

$$(1 * 100 * wad)/(10,000 * wad) = 0.01$$

Since the dt is being converted to u64, the fractional part is truncated, resulting in a value of 0 for dt.

Code Location:

Down below is the code snippet from the update_rewards_batch_speed function:

Listing 2: controller/src/governance.rs

```
331 let dt = match BigUint::to_u64(&new_dt) {
332     None => sc_panic!(ERROR_UNEXPECTED_REWARDS_BATCH_PERIOD),
333     Some(dt) => dt,
334 };
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:L/A:L/D:N/Y:N/R:N/S:U (3.1)

Recommendation:

To mitigate this issue, it is recommended to add an explicit check for a zero value of `dt` and handle it appropriately. This might include returning an error or requiring a minimum rewards batch period.

Remediation Plan:

SOLVED: The issue was solved in commit a76fa by adding a check to verify the reward's period is greater than zero when updating rewards speed.

3.3 (HAL-03) SINGLE STEP OWNERSHIP TRANSFER PROCESS - LOW (3.3)

Description:

The current ownership transfer process involves the approval of a proposal. When the proposal is successful, the admin is set to the proposed address in the try_change_admin function. This function checks if the new owner is not the zero address and proceeds to write the new admin. If the submitted account is not a valid account, it is entirely possible the owner may accidentally transfer ownership to an uncontrolled account, breaking the access control in the execute endpoint and in the proposals' creation process. Lack of a two-step procedure for critical operations leaves them error-prone if the address is incorrect, the new address will take the administration of the contract.

Code Location:

Down below is the code snippet from the try_change_admin function:

Listing 3: governance/src/config.rs (Line 85)

```
82 fn try_change_admin(&self, address: ManagedAddress) {
83         require!(!address.is_zero(), ERROR_ADDRESS_ZERO);
84
85         self.admin().set(&address);
86     }
```

BVSS:

AO:A/AC:L/AX:H/C:N/I:N/A:C/D:N/Y:N/R:N/S:U (3.3)

Recommendation:

It's recommended to introduce a validation step to confirm the new administrator address before updating the admin field, for example, requiring

the new administrator to accept the role explicitly, potentially through a function call from the new admin address.

Remediation Plan:

SOLVED: This issue was solved in commit 1804e by requiring the new admin to accept their role to finalize the role transfer. It is important to note that the governance contract was moved to a new repository.

3.4 (HAL-04) ANCHOR TOLERANCE DOES NOT HAVE A THRESHOLD IN THE ORACLE CONSTRUCTOR - INFORMATIONAL (0.0)

Description:

The anchor tolerance is set to ensure the acceptable deviation between the oracle's reported price and the actual market price of the asset. In the oracle's `init` function, this deviation is set using the parameter `anchor_tolerance`. The function `set_anchor_tolerance_internal` checks that the lower_bound is at least 1, but the upper bound is 1 + the submitted anchor tolerance, so the upper bound can have any value.

Big upper bounds could make the oracle vulnerable to price manipulations and/or high volatility.

Code Location:

Down below is the code snippet from the `set_anchor_tolerance_internal` function:
The upper bound is calculated here:

Listing 4: oracle/src/common.rs (Line 40)

```
38 fn set_anchor_tolerance_internal(&self, anchor_tolerance: &BigUint
↳ ) {
39         let wad = BigUint::from(WAD);
40         let upper_bound = &wad + anchor_tolerance;
41         let lower_bound = if anchor_tolerance < &wad {
42             &wad - anchor_tolerance
43         } else {
44             // avoid prices being zero
45             BigUint::from(1u32)
46         };
47         self.upper_bound_ratio().set(upper_bound);
48         self.lower_bound_ratio().set(lower_bound);
49     }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

It is recommended to set a max value for the upper bound, so the anchor tolerance is always between a reasonable value.
t’

Remediation Plan:

SOLVED: The issue was solved in commit 035ec by setting a max_anchor_tolerance value as threshold.

3.5 (HAL-05) REDUNDANT CHECK FOR UNDERLYING AMOUNT GREATER THAN ZERO - INFORMATIONAL (0.0)

Description:

In the `borrow_internal` function, there is a redundant check for `underlying_amount > BigUint::zero()`. This check is unnecessary as the `borrow` function, which calls `borrow_internal`, has already ensured that the `underlying_amount` is greater than zero. Removing this redundant check can improve code efficiency.

Code Location:

Down below is the code snippet from the `borrow_internal` function:

Listing 5: /money-market/src/borrow.rs

```
48 if borrower_current_borrow_amount == BigUint::zero() &&
↳ underlying_amount > BigUint::zero() {
49     self.borrower_count().update(|_count| *_count += 1u64);
50 }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

To address this issue, it is recommended to remove the unnecessary check for `underlying_amount > BigUint::zero()` in the `borrow_internal` function.

Remediation Plan:

SOLVED: The issue was solved in the commit 47658 by removing the `underlying_amount > BigUint::zero()` check.

3.6 (HAL-06) MISSING INTEREST RATE MODEL VALIDATION - INFORMATIONAL (0.0)

Description:

The get_interest_rate_model_proxy function does not validate if the interest rate model address is defined before accessing it, which may lead to unexpected behavior if the address is not set, or has been accidentally removed or invalidated.

Code Location:

Down below is the code snippet from the get_interest_rate_model_proxy function:

Listing 6: /money-market/src/proxies.rs

```
118 i    fn get_interest_rate_model_proxy(&self, sc_address: Option<
 ↳ ManagedAddress>) -> interest_rate_model::Proxy<Self::Api> {
119         match sc_address {
120             Some(interest_rate_model_address) => self.
 ↳ interest_rate_model_proxy(interest_rate_model_address),
121             None => {
122                 let interest_rate_model_address = self.
 ↳ interest_rate_model().get();
123                 self.interest_rate_model_proxy(
 ↳ interest_rate_model_address)
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

To mitigate this vulnerability, it is recommended to add a check before accessing the interest rate model to ensure that it is defined and valid.

Remediation Plan:

SOLVED: The issue was solved in the commit ac0fc by adding a check to verify if the interest rate model contract has been defined.

3.7 (HAL-07) SUPPORTED PAIRS CANNOT BE MADE UNSUPPORTED - INFORMATIONAL (0.0)

Description:

It was observed that it is not possible to remove a supported pair from the `whitelisted_maiar_pairs` nor the `whitelisted_chainlink_pairs`. The only way to remove those pairs is overwriting its value, or pausing the market.

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

It is recommended to write a function to empty values from the `whitelisted_maiar_pairs` or to pause the use of supported pairs.

Remediation Plan:

SOLVED: The issue was solved in commit `1eacfc` by introducing the `removeToken` endpoint, which deletes all information related to a previously supported Token. The Hatom protocol now places more emphasis on the token itself, rather than on pairs.

MANUAL TESTING

4.1 INTEREST RATE MODEL

- The interest rate calculation is accurate under normal conditions. - PASSED
- The interest rate calculation handles edge cases correctly, such as zero or extremely high utilization rates - PASSED
- The interest rate model can be updated by an authorized admin or governance action - PASSED
- Unauthorized users cannot update the interest rate model - PASSED

4.2 MONEY MARKET

- Users can borrow and repay loans according to the platform's rules - PASSED
- Accrual of interest for borrowers and suppliers is updated with frequency - PASSED
- Liquidations work correctly when collateral value falls below the required threshold - PASSED
- Users cannot manipulate the system to obtain loans exceeding their allowed borrowing capacity - PASSED

4.3 ORACLE

- The oracle returns accurate price data for supported assets - PASSED
- Oracle's response to extreme price fluctuations or unexpected asset price data can be abused to steal funds - PASSED

4.4 CONTROLLER

- Controller enforces proper collateralization and borrowing limits for users - PASSED

- Controller accurately calculates interest rates and updates user balances accordingly - PASSED
- Controller's handling of user actions, such as depositing collateral, borrowing, repaying loans, and withdrawing collateral can be abused to obtain profits - PASSED
- Controller responds correctly to external factors, such as changes in the interest rate model or oracle price data - PASSED

4.5 ADMIN

- Admin actions are restricted to authorized admin accounts - PASSED
- Admin actions, such as updating interest rate models or adding new collateral types, function as expected - PASSED
- Unauthorized users cannot perform admin actions - PASSED

4.6 GOVERNANCE

- Governance system enforces proper voting weights and quorum requirements - PASSED
- Governance proposals result in the desired changes to the protocol - PASSED
- Users cannot manipulate the governance process or bypass voting requirements - PASSED

AUTOMATED TESTING

5.1 AUTOMATED ANALYSIS

Description:

Halborn used automated security scanners to assist with detection of well-known security issues and vulnerabilities. Among the tools used was cargo audit, a security scanner for vulnerabilities reported to the RustSec Advisory Database. All vulnerabilities published in https://crates.io are stored in a repository named The RustSec Advisory Database. cargo audit is a human-readable version of the advisory database which performs a scanning on Cargo.lock. Security Detections are only in scope. To better assist the developers maintaining this code, the auditors are including the output with the dependencies tree, and this is included in the cargo audit output to better know the dependencies affected by unmaintained and vulnerable crates.

ID	package	Short Description
RUSTSEC-2022-0054	wee_alloc 0.4.5	wee_alloc is unmaintained

Listing 7: Dependency tree

```
wee_alloc 0.4.5
 elrond-wasm 0.34.1
    staking 0.0.0
       hatom-protocol 0.0.0
    safety-module 0.0.0
       hatom-protocol 0.0.0
    pair 0.0.0
       hatom-protocol 0.0.0
    oracle-mock 0.0.0
       hatom-protocol 0.0.0
    oracle 0.0.0
       hatom-protocol 0.0.0
       controller 0.0.0
           money-market 0.0.0
              staking 0.0.0
              safety-module 0.0.0
              hatom-protocol 0.0.0
```

```
18            hatom-protocol 0.0.0
19     money-market 0.0.0
20     interest-rate-model 0.0.0
21        money-market 0.0.0
22        hatom-protocol 0.0.0
23     hatom-protocol 0.0.0
24     governance 0.0.0
25        staking 0.0.0
26        safety-module 0.0.0
27        hatom-protocol 0.0.0
28     elrond-wasm-debug 0.34.1
29        hatom-protocol 0.0.0
30     controller 0.0.0
31     chainlink-mock 0.0.0
32        hatom-protocol 0.0.0
33     admin 0.0.1
34         staking 0.0.0
35         safety-module 0.0.0
36         oracle 0.0.0
37         money-market 0.0.0
38         hatom-protocol 0.0.0
39         controller 0.0.0
40  elrond-codec 0.12.0
41      elrond-wasm 0.34.1
```

THANK YOU FOR CHOOSING

// HALBORN