Civitia

Audit Report

MOVEBIT

contact@bitslab.xyz

https://twitter.com/movebit_

Fri Jun 27 2025

Civitia Audit Report

1 Executive Summary

1.1 Project Information

Description	Civitia is an onchain game with mechanics similar to Monopoly that invites users to collaborate and interact within a unique social and financial ecosystem. It is built on an independent rollup with the Initia stack.
Type	Game
Auditors	MoveBit
Timeline	Wed May 28 2025 - Mon Jun 09 2025
Languages	Move
Platform	Others
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/civitia-labs/civitia-contracts
Commits	73fb65fcf2b6a585ba251068ba4528583ab40bdd a20a72b53179ee3ce444200afbbab4baca1886ad

1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
CIV	sources/civitia.move	b6c3f8f0c380f661bb1951e45882c41410c7aa4d
LOB	sources/lobby.move	de0d327c215cf66bd730164f234e6d403b2dd0a2
CIT	sources/city.move	f568411743490ed61c768aff47b4aea35f6b06a6
RV2	sources/random_v2.move	366574afcd392b61b33eda8296a127852904da7a
SQU	sources/square.move	79920457f25eba84b5ac72ac1aa1541393894272
EPO	sources/epoch.move	42229628bd67ca4f7b741aa60ae303c77b200e21
FCN	sources/founding_citizens_nfts.move	5d5c745bb77ca636f52dbbf25477c288ee41feff
RES	sources/residence.move	ae327733dfb5f5f8a074992503a3e871c4f2fe2a
BOA	sources/board.move	c455df20242e7c76be4ea867eb5ac870362d2d15
CON	sources/config.move	156e0a9bc89408442d44273d96b4371ca9c9f71a

PLA	sources/player.move	8b0937c288d1f24e39046846353b93e054318ad7
BUN	sources/bunker.move	694091f66a8a6221c2cee9d04ffc42a9b83e1798
JAI	sources/jail.move	c491287112557539fbfed40f56aaa733fba6e008
TAU	sources/tax_authority.move	a7fba61220e70bfa6a8ad2be689aded1fa1a10b7
WCI	sources/whitelist_city.move	d761bad555d56550929732ffc92670f4ee0b1768
RAN	sources/random.move	09ed62e368a2a8b6b5412d42bfa571da6f5e64f2
EGU	sources/entry_guard.move	e0a935213509652f3fa36d1324b8d18cccf00d0a
VIP	sources/vip.move	8665ea2984a6afc4b5986ff73869f034cdf10add
SEA	sources/seasons.move	3d1d14ac89d1ba426ffe476f85ef1921e89b9de7
TRE	sources/treasury.move	897cbe98132c3b2ffd80a445b5c77e247b9750d6
CCA	sources/citizen_cards.move	eeff098549d89ae876a0a27a930582a079d561e0
LEV	sources/levels.move	3b495832e308498c5c5a90e19e98e1413298a558

1.3 Issue Statistic

Item	Count	Fixed	Partially Fixed	Acknowledged
Total	8	3	1	4
Informational	5	2	0	3
Minor	1	0	0	1
Medium	1	0	1	0
Major	1	1	0	0
Critical	0	0	0	0

1.4 MoveBit Audit Breakdown

MoveBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow by bit operations
- Number of rounding errors
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting
- Unchecked CALL Return Values
- The flow of capability
- Witness Type

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** and **"Formal Verification"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Formal Verification(Optional)

Perform formal verification for key functions with the Move Prover.

(4) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

This report has been commissioned by Civitia to identify any potential issues and vulnerabilities in the source code of the Civitia smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we identified 8 issues of varying severity, listed below.

ID	Title	Severity	Status
BOA-1	Incorrect Start Point Detection Logic	Major	Fixed
EGU-1	Lack of Event Emit	Informational	Acknowledged
LEV-1	Incorrect Maximum Level Validation in `is_max_level` Function	Informational	Acknowledged
RV2-1	Mixed Test and Production Code	Minor	Acknowledged
TRE-1	Centralization Risk	Medium	Partially Fixed
TRE-2	Improper Function Visibility in `Treasury` Withdrawal	Informational	Acknowledged
TRE-3	Redundant Metadata Validation in `Treasury` Deposit Function	Informational	Fixed
VIP-1	Missing Initialization Check in `finalize_stage` Function	Informational	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the Civitia Smart Contract :
Owner

- The owner can call the `create_city_square` function to create new city squares on the board.
- The owner can call the `create_bunker_square` function to create new bunker squares.
- The owner can call the `create_tax_authority_square` function to create new tax authority squares.
- The owner can call the `create_whitelist_city_square` function to create new whitelist city squares.
- The owner can call the `create_jail_square` function to create new jail squares.
- The owner can call the `create_citizen_cards_square` function to create new citizen cards squares.
- The owner can call the `create_receive_silver_per_unit_card` function to create silver reward cards.
- The owner can call the `create_receive_tp_card` function to create TP reward cards.
- The owner can call the `create_move_to_square_card` function to create movement cards.
- The owner can call the `create_pay_to_jackpot_card` function to create jackpot payment cards.
- The owner can call the `create_pay_to_jackpot_per_unit_card` function to create per-unit jackpot payment cards.
- The owner can call the `set_board_squares_by_id` function to configure the board layout.
- The owner can call the `set_radiation_levels` function to configure city radiation parameters.

- The owner can call the `set_landlord_levels` function to configure landlord level thresholds and multipliers.
- The owner can call the `set_city_mint_fee` function to update city unit minting fees.
- The owner can call the `set_city_burn_fee` function to update city unit burning fees.
- The owner can call the `set_bunker_burn_fee` function to update bunker burning fees.
- The owner can call the `set_whitelist_city_base_rent` function to update whitelist city rent amounts.
- The owner can call the `set_jail_config` function to update jail configuration parameters.
- The owner can call the `set_tax_authority_config` function to update tax authority settings.
- The owner can call the `set_residence_pass_price` function to update residence pass pricing.
- The owner can call the `set_season_reward_distribution_for_top_positions` function to configure season reward distribution.
- The owner can call the `set_seasons_duration_in_epochs` function to set season duration.
- The owner can call the `set_current_season_end_epoch` function to set the current season end time.
- The owner can call the `set_fees_receiver` function to update the fees receiver address.
- The owner can call the `set_game_master` function to transfer ownership to a new game master.
- The owner can call the `set_vip_stage_manager` function to set the VIP stage manager address.
- The owner can call the `set_is_halted` function to halt or resume game operations.
- The owner can call the `set_lobby_params` function to configure lobby parameters.

- The owner can call the `set_is_sale_active` function to activate or deactivate NFT sales.

User

- Users can call the `initialize_player` function to create their player account with a referrer.
- Users can call the `roll_dice` function to move across the game board.
- Users can call the `mint_current_city_unit` function to purchase city units when standing on city squares.
- Users can call the `burn_city_units` function to sell their owned city units.
- Users can call the `claim_city_rents` function to collect rent rewards from owned city units.
- Users can call the `sabotage_city` function to use Tax Points to negatively impact city scores.
- Users can call the `buy_and_establish_residence` function to purchase and establish residency in cities.
- Users can call the `claim_whitelist_city_rents` function to collect rent from whitelist city units.
- Users can call the `claim_all_rents` function to collect all available rent rewards.
- Users can call the `mint_current_bunker` function to purchase bunkers when standing on bunker squares.
- Users can call the `claim_bunker_rents` function to collect rent rewards from owned bunkers.
- Users can call the `sabotage_bunker` function to use Tax Points to damage bunkers.
- Users can call the `file_current_ta_taxes` function to file taxes when on Tax Authority squares.
- Users can call the `pay_current_jail_bail` function to pay bail when in jail.
- Users can call the `bribe_current_jail` function to attempt bribing their way out of jail.

- Users can call the `claim_season_rewards` function to claim rewards from completed seasons.
- Users can call the `donate_to_current_season_jackpot` function to contribute to the season jackpot.
- Users can call the `draw_citizen_card` function to draw cards when on citizen card squares.
- Users can call the `whitelist_v2` function to whitelist their account during the lobby phase.
- Users can call the `mint_capsules` function to mint Founding Citizens Capsule NFTs.
- Users can call the `mint_orbs` function to mint Founding Citizens Orb NFTs.
- Users can call the `mint_tanks` function to mint Founding Citizens Tank NFTs.

4 Findings

BOA-1 Incorrect Start Point Detection Logic

Severity: Major

Status: Fixed

Code Location:

sources/board.move#198

Descriptions:

The current implementation uses `new_square_index < player_current_square_index` to detect if a player has passed the starting point after rolling dice. However, this logic fails to account for cases where the player's movement results in a full loop around the board (e.g., when `(player_current_square_index + rolled_number) % board.squares_size == 0`). In such scenarios, the player returns to the exact starting position without triggering the "pass go" condition, potentially allowing them to bypass rewards or actions tied to completing a full circuit.

Suggestion:

It is recommended to replace the comparison logic with `(player_current_square_index + rolled_number) >= board.squares_size` to accurately detect all cases where the player completes a full loop.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

EGU-1 Lack of Event Emit

Severity: Informational

Status: Acknowledged

Code Location:

sources/entry_guard.move#50

Descriptions:

Functions such as `set_is_halted()` , `set_lobby_params` lack logs, making the contract's activities difficult to track.

Suggestion:

It is recommended to add event emission for this operation.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

LEV-1 Incorrect Maximum Level Validation in `is_max_level` Function

Severity: Informational

Status: Acknowledged

Code Location:

sources/levels.move#118

Descriptions:

The current implementation of `is_max_level` uses the condition `vector::length(&config.levels) <= level + 1`, which incorrectly identifies invalid levels (e.g., `level > length`) as maximum levels. This a logical inconsistency where out-of-bound indices are treated as valid maximum levels, even though they do not exist in the configuration.

Suggestion:

It is recommended to modify the condition to explicitly check if `level` less than `length - 1`.

RV2-1 Mixed Test and Production Code

Severity: Minor

Status: Acknowledged

Code Location:

sources/random_v2.move

Descriptions:

The `PredeterminedRandom` struct and its associated test logic are designed for testing purposes but remain present in production code paths. While test-only attributes (`#[test_only]`) prevent direct usage in production, the conditional checks (e.g., `exists<PredeterminedRandom>(@civitia)`) still introduce unnecessary branching logic into the final compiled bytecode.

Suggestion:

It is recommended to remove the test-only conditionals from production functions like `rand_u64` and `rand_u64_range` .

TRE-1 Centralization Risk

Severity: Medium

Status: Partially Fixed

Code Location:

sources/treasury.move#21

Descriptions:

Centralization risk was identified in the smart contract:

- Admin can withdraw any store created by `create_treasury`.

Suggestion:

It is recommended that measures be taken to reduce the risk of centralization, such as a multi-signature mechanism.

Resolution:

The client replied that: The deployment of all civitia modules is done in fact by a multisig, called Civitia DAO. More details about Civitia DAO can be found in the docs https://docs.civitia.org/community/civitia-dao Therefore, no single actor can have access to the stores created by treasury module.

TRE-2 Improper Function Visibility in `Treasury` Withdrawal

Severity: Informational

Status: Acknowledged

Code Location:

sources/treasury.move#41

Descriptions:

The current implementation exposes the `withdraw` function with public visibility, creating a vulnerability if any future upgrade introduces public functions that return or expose the `Treasury` struct instance.

Suggestion:

It is recommended to restrict the function visibility by:

1. Changing to `public(friend)` and explicitly declaring trusted modules in the `friend` list.
2. Or using `internal` visibility with controlled access through module-internal dispatchers.

TRE-3 Redundant Metadata Validation in `Treasury` Deposit Function

Severity: Informational

Status: Fixed

Code Location:

sources/treasury.move#51

Descriptions:

The current implementation performs duplicate metadata validation in the `deposit` function:

1. Explicit check `asset_metadata == coin_metadata(treasury)` before deposit
2. Implicit check within `fungible_asset::deposit` (which already validates metadata internally).

Suggestion:

It is recommended to remove the external metadata check to rely solely on the internal validation.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

VIP-1 Missing Initialization Check in `finalize_stage` Function

Severity: Informational

Status: Fixed

Code Location:

sources/vip.move

Descriptions:

The `finalize_stage` function directly uses `store.stage` without verifying whether the VIP module has been initialized (`is_vip_initialized`). In contrast, `increase_score` includes a safety check for `is_vip_initialized` .

Suggestion:

It is recommended to add an explicit check for initialization status in `finalize_stage` .

Resolution:

This issue has been fixed. The client has adopted our suggestions.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer