CityCoins Audit

December 2021

By CoinFabrik

Introduction

CoinFabrik was asked to audit the contracts for the CityCoins project. First we will provide a summary of our discoveries and then we will show the details of our findings.

Scope

The contracts audited are from the https://github.com/citycoins/citycoin repository. The audit is based on the commit `653056fe4f58b4eed5674009c55cedf0dc2dced3`. Fixes were checked on `9dbc51e80e2653bd3d77175cd44c3e411ce2d7c9`.

The audited contracts are:

- `contracts/citycoin-vrf.clar`: Contains the required functions to generate an integer from the specified block's VRF seed.
- `contracts/citycoin-core-trait.clar`: Trait definition for a core contract.
- `contracts/citycoin-token-trait.clar`: Trait definition for a token contract.
- `contracts/MiamiCoin/auth.clar`: Administrative functions for the approvers and city wallet roles.
- `contracts/MiamiCoin/token.clar`: Token implementation.
- `contracts/MiamiCoin/core-v1.clar`: Contains mining, staking and reward claiming functions.

Analyses

The following analyses were performed:

- Misuse of the different call methods
- Integer overflow errors
- Division by zero errors
- Front running attacks
- Reentrancy attacks
- Misuse of block timestamps
- Softlock denial of service attacks
- Functions with excessive gas cost
- Missing or misused function qualifiers
- Needlessly complex code and contract interactions
- Poor or nonexistent error handling
- Failure to use a withdrawal pattern
- Insufficient validation of the input parameters
- Incorrect handling of cryptographic signatures

Findings and Fixes

ID	Title	Severity	Status
CR-01	Lost Miners Funds after Shut Down	Critical	Acknowledged
ME-01	Insecure Authentication through tx-sender	Medium	Mitigated
MI-01	Rounding Issue when Calculating Stacking Rewards	Minor	Acknowledged
MI-02	Inactive Core Contract Can Activate Itself	Minor	Fixed
MI-03	Malicious Inactive Core Contract Can Mint and Burn Tokens	Minor	Acknowledged
MI-04	Denial of Service in the Upgrade of a Malicious Core	Minor	Mitigated
EN-01	Approvers Need to Approve his Own Jobs	Enhancement	Not fixed
EN-02	Inactive Core Contract Can be Target of an Upgrade	Enhancement	Fixed

Severity Classification

Security risks are classified as follows:

- **Critical:** These are issues that we manage to exploit. They compromise the system seriously. They must be fixed **immediately**.
- **Medium:** These are potentially exploitable issues. Even though we did not manage to exploit them or their impact is not clear, they might represent a security risk in the near future. We suggest fixing them **as soon as possible**.
- **Minor:** These issues represent problems that are relatively small or difficult to take advantage of but can be exploited in combination with other issues. These kinds of issues do not block deployments in production environments. They should be taken into account and be fixed **when possible**.
- **Enhancement:** These kinds of findings do not represent a security risk. They are best practices that we suggest to implement.

This classification is summarized in the following table:

SEVERITY	EXPLOITABLE	ROADBLOCK	TO BE FIXED
Critical	Yes	Yes	Immediately
Medium	In the near future	Yes	As soon as possible
Minor	Unlikely	No	Eventually
Enhancement	No	No	Eventually

Issues Found by Severity

Critical Severity Issues

CR-01 Lost Miners Funds after Shut Down

In the core contract, users can commit to mine as many as the next 200 blocks. The city wallet or the approvers can upgrade the core contract through the auth contract (`auth.clar`), stopping the mining in the previous core. Therefore, those miners who committed for the blocks after the upgrade will have lost their committed tokens.

Recommendation

Implement a function to refund miners which is called by the shutdown function (`shutdown-contract()`). This function should store the committed values in a mapping. Finally, miners would be able to claim their tokens by calling a withdrawal function.

Status

Acknowledged. Additional research will be done on this issue. In the meantime, the development team expressed that upgrades would only be performed when absolutely necessary, scheduled in advance and well-communicated.

Medium Severity Issues

ME-01 Insecure Authentication through tx-sender

Global variable `tx-sender` returns the original sender of the current transaction, or if `as-contract` was called to modify the sending context, it returns that contract principal. Using this variable for authentication is not secure. Actors in the system could be targets of phishing. This is analogous to what happens to `tx.origin` and `msg.sender` in Solidity. There, the convention is to use `msg.sender`, which works like `contract-caller`.

For instance, an approver can be tricked to call a malicious contract which calls `auth.approve-job()` against his will. Also, the malicious contract could call the token transfer function (`token.transfer()`) to send the user's tokens to itself.

Recommendation

Prefer `contract-caller` to `tx-sender` for authentication. `contract-caller` returns the caller of the current contract context.

Status

Mitigated. Job approval flow requires either two-of-three multi-signature transactions or three-of-five approvers' votes.

Minor Severity Issues

MI-01 Rounding Issue when Calculating Stacking Rewards

In the core contract, stacking rewards are calculated through `get-entitled-stacking-reward()`. The amount of city coins locked by the user is multiplied by the total amount of micro-stacks staked for a cycle, and then divided by the total amount of city coins stacked for that cycle. Because of the data type imprecision, remainders will be accumulated in the contract and will not be used.

Recommendation

If there were a considerable amount of micro-stacks over time, the contract could calculate this value and use it. When all the stackers have claimed their rewards, the difference between the claimed amount and `totalUstxThisCycle` is the sum of these remainders.

Status

Acknowledged. The accumulated value is not considered worth enough to assume the runtime and implementation additional costs.

MI-02 Inactive Core Contract Can Activate Itself

All core contracts are stored in the auth contract with different states: `STATE_DEPLOYED, STATE_ACTIVE` and `STATE_INACTIVE`. The deployed state is the initial state. When it receives the required signals, it transits to the active state. Finally, if it is upgraded, the previous contract will be inactive. `auth.activate-core-contract()` only checks if the caller is in the core contract mapping. If it were a malicious contract which was replaced through an upgrade, it would call `activate-core-contract()` and activate itself.

Recommendation

Verify the contract caller is a core registered with state `STATE_DEPLOYED`.

Status

Fixed. The recommendation was implemented.

MI-03 Malicious Inactive Core Contract Can Mint and Burn Tokens

The token contract (`token.clar`) only validates if the contract caller is registered in the auth contract. However, contracts cannot be removed from the auth contract. Therefore, if a malicious core were added and then replaced by the approvers, it could still mint new tokens and burn someone else's tokens.

Recommendation

Instead of validating if the core contract is registered, the token contract should check if the core is active through `get-active-core-contract()`.

Status

Acknowledged. This was a design choice. Core contracts need to be able to mint in order to reward the miners who have not claimed their tokens before the upgrade. Also, the unintended inclusion of a malicious core contract is mitigated by the job approval flow requirements (either two-of-three multi-signature transactions or three-of-five approvers' votes).

MI-04 Denial of Service in the Upgrade of a Malicious Core

`upgrade-core-contract()` and `execute-upgrade-core-contract-job()` are the functions that set a new active core contract. These functions call `shutdown-contract()` from the current active core to disable it. However, if the current core is a malicious contract, it might revert to stop the upgrade, making it impossible to change the core.

Recommendation

Instead of calling the core contract to modify the variables, those variables could be placed in the auth contract and be checked by the core through external calls. Therefore, the upgrade does not depend on unknown external logic.

Status

Mitigated. The unintended inclusion of a malicious core contract is mitigated by the job approval flow requirements (either two-of-three multi-signature transactions or three-of-five approvers' votes).

Enhancements

EN-01 Approvers Need to Approve his Own Jobs

In the auth contract (`auth.clar`), approvers can create a job. For a job to be executed, it requires a minimum amount of approvals. Now, if an approver creates a job through `create-job()`, he can call `approve-job()` to approve his own job.

Recommendation

The job creation function should also add the creator approval. Then, the job creator does not need to make another transaction to approve his own job.

Status

Not fixed.

EN-02 Inactive Core Contract Can be Target of an Upgrade

In the auth contract (`auth.clar`), `upgrade-core-contract()` and `execute-upgrade-core-contract-job()` does not check if the new core contract was already registered before. Therefore, an inactive core can be passed as an argument and then be activated. This would result in a poor experience because that core was shut down when it was replaced and most of the functions are disabled.

Recommendation

Validate the new contract is not already registered in the auth contract.

Status

Fixed.

Other Considerations

- The burn function (`token.burn()`) allows a core contract to burn users' tokens without their consent. The audited core contract does not implement any function which calls `token.burn()`, but a new core contract can be included and call that function.
  The new commit modified the `burn()` function, allowing only the owners to burn their own tokens.

Conclusion

We found the contracts to be simple and straightforward and have an adequate amount of documentation. We found a critical issue, a medium issue and several minor issues and enhancements.

An issue was fixed, two were mitigated and three were acknowledged.

Disclaimer: This audit report is not a security warranty, investment advice, or an approval of the CityCoins project since CoinFabrik has not reviewed its platform. Moreover, it does not provide a smart contract code faultlessness guarantee.