KannaCoin Audit

November 2022

By CoinFabrik

v202209

Executive Summary

CoinFabrik was asked to audit the contracts for the KannaCoin project. The audited files are from the git repository located at https://github.com/kanna-coin/treasury-staking-monorepo. The audit is based on the commit `4d00e8dd08ce7532f4697fdd0a177110713eedec`.

During this audit, we found a critical issue and two minor issues. Also, several enhancements were proposed.

Scope

The scope for this audit includes and is limited to the following files:

- `contracts/KannaPreSale.sol`: Pre-sale contract for Kanna.
- `contracts/KannaToken.sol`: ERC20 token.
- `contracts/KannaTreasurer.sol`: Vault contract for Kanna.
- `contracts/KannaYield.sol`: Kanna farming contract.

No other files in this repository were audited. Its dependencies are assumed to work according to their documentation. Also, no tests were reviewed for this audit.

Methodology

CoinFabrik was provided with the source code, including automated tests that define the expected behavior, and general documentation about the project. Our auditors spent one week auditing the source code provided, which includes understanding the context of use, analyzing the boundaries of the expected behavior of each contract and function, understanding the implementation by the development team (including dependencies beyond the scope to be audited) and identifying possible situations in which the code allows the caller to reach a state that exposes some vulnerability. Without being limited to them, the audit process included the following analyses:

- Arithmetic errors
- Outdated version of Solidity compiler
- Race conditions
- Reentrancy attacks
- Misuse of block timestamps
- Denial of service attacks
- Excessive gas usage
- Missing or misused function qualifiers
- Needlessly complex code and contract interactions
- Poor or nonexistent error handling

- Insufficient validation of the input parameters
- Incorrect handling of cryptographic signatures
- Centralization and upgradeability

Findings

In the following table we summarize the security issues we found in this audit. The severity classification criteria and the status meaning are explained below. This table does not include the enhancements we suggest to implement, which are described in a specific section after the security issues.

ID	Title	Severity	Status
CR-01	Inconsistent State And Denial Of Service After Claiming	Critical	Unresolved
MI-01	Contract Initialized With Zero-address	Minor	Unresolved
MI-02	Overflow Because Of Casting	Minor	Unresolved

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 might 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**.

Issues Status

An issue detected by this audit has one of the following statuses:

- **Unresolved**: The issue has not been resolved.
- **Acknowledged**: The issue remains in the code, but is a result of an intentional decision.
- **Resolved**: Adjusted program implementation to eliminate the risk.

- **Partially resolved**: Adjusted program implementation to eliminate part of the risk. The other part remains in the code, but is a result of an intentional decision.
- **Mitigated**: Implemented actions to minimize the impact or likelihood of the risk.

Critical Severity Issues

CR-01 Inconsistent State And Denial Of Service After Claiming

Location:

- `contracts/KannaPreSale.sol:145-161`

When `claim()` is called with `unlock=false`, it is not supposed to use locked funds. However, the function does not verify if the contract has enough tokens in the unlocked balance in order to transfer to the claimer. Therefore, it will use locked funds and not update the internal balance related to the locked tokens, resulting in an erroneous state.

Exploiting it, besides using locked funds when explicitly expressed the opposite, will result in `availableSupply()` reverting because of an overflow. Since most of the functions depend on it, it will cause a denial of service.

Recommendation

Add a `require` statement to check if the available supply is greater than the claimed amount when `unlock=false`.

Status

Unresolved.

Medium Severity Issues

No issues found.

Minor Severity Issues

MI-01 Contract Initialized With Zero-address

Location:

- `contracts/KannaPreSale.sol:51-61,`
- `contracts/KannaYield.sol:55-63`

In `KannaPreSale`, the constructor function does not check whether knnToken address is different from zero, what will result in functions reverting because of failed external calls. Moreover, knnToken is an immutable variable, then it cannot be changed afterwards.

In `KannaYield`, the same happens to `feeRecipient`, which also is an immutable variable. In this case, this results in transferring collected fees to the zero-address.

Recommendation

Add a check to verify the set addresses are not zero.

Status

Unresolved.

MI-02 Overflow Because Of Casting

Location:

- `contracts/KannaPreSale.sol:203`

In the `convertToWEI()` function, `answer`, an `int256` variable, is cast to `uint256`. If the variable has a value lower than zero, `ethPriceInUSD` will overflow and wrap, resulting in a large value.

Since the price aggregator is out of the scope of the audit, the likelihood of this issue is unknown.

Recommendation

Use the `safeCast` library included in OpenZeppelin's module.

Status

Unresolved.

Enhancements

These items do not represent a security risk. They are best practices that we suggest implementing.

ID	Title	Status
EN-01	Declare Compile-Time-Known Values As Constants	Not implemented
EN-02	Unnecessary Cast	Not implemented
EN-03	Unnecessary Zero-address Check	Not implemented

EN-01 Declare Compile-Time-Known Values As Constants

Location:

- `contracts/KannaToken.sol:24,25,`
- `contracts/KannaYield.sol:53`

In `KannaToken`, INITIAL_SUPPLY and MAX_SUPPLY are immutable variables set inline. However, they can be declared as constants, what will save gas.

Also, in `KannaYield`, `subscriptionFee` is a variable which value is set when deployed and it is not changed afterwards.

Recommendation
Declare those values as constants.

Status
Not implemented.

EN-02 Unnecessary Cast

Location:

- `contracts/KannaToken.sol:57`

The parameter newTreasury is an address, and it is cast to address, which is redundant and increases runtime cost.

Recommendation
Remove unnecessary cast.

Status
Not implemented.

EN-03 Unnecessary Zero-address Check

Location:

- `contracts/KannaToken.sol:72`

The `mint()` function verifies if `treasury` address is equal to zero. However, `_mint()` already verifies it.

Recommendation
Remove unnecessary check in order to reduce runtime cost.

Status
Not implemented.

Other Considerations

The considerations stated in this section are not right or wrong. We do not suggest any action to fix them. But we consider that they may be of interest to other stakeholders of the project, including users of the audited contracts, token holders or project investors.

Centralization

For KannaPreSale, the owner can set price for the tokens, withdraw the funds, assign roles to other addresses and stop the presale.

For KannaToken, the owner can update the treasury, add or remove minters,

The owner of KannaTreasurer can withdraw the tokens from the vault.

For KannaYield, the owner should transfer the tokens used to reward users. Otherwise, no rewards will be distributed.

Upgrades

Contracts does not provide mechanisms for upgrades.

Privileged Roles

These are the privileged roles that we identified on each of the audited contracts.

Owner Can Set Any Price For Presale

The function `updateQuotation()` allows the contract owner to set the price at which KannaCoin is bought.

Use Of Transfer Method For Sending Ether

`KannaPreSale::withdraw()` sends ether through the `transfer()` method, which is limited in the gas which can be used by the recipient. Therefore, if the recipient is a contract with a `fallback()`/`receive()` function that has a high runtime cost, it will revert.

Tier List Should Be Ordered

`KannaYield::feeOf()` iterates over the `tier` list until `subscriptionDuration` is lower than one of the values and returns the corresponding fee. However, if the list is unordered, a greater incorrect state might be returned.

Changelog

- 2022-11-11 – Initial report based on commit `4d00e8dd08ce7532f4697fdd0a177110713eedec`.

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