Smart contract security audit FermionToken

v.1.0

Table of Contents

1.0 Introduction

1.1 Project engagement

During January of 2022, FermionToken engaged CTDSec to audit smart contracts that they created. The engagement was technical in nature and focused on identifying security flaws in the design and implementation of the contracts. FermionToken provided CTDSec with access to their code repository and whitepaper.

Fermion token is an innovative project that rewards purchase orders with a bonus mechanism using onchain/offchain oracles.

1.2 Disclaimer

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that’s able to withstand the network’s fast-paced and rapidly changing environment, we at CTDSec recommend that FermionToken team put in place a bug bounty program to encourage further and active analysis of the smart contract.

Owner wallets should also be multisig to mitigate the risks of centralization. Wallet signatories should be identified and have appropriate authentication and identity confirmation prior to implementation.

2.0 Coverage

2.1 Target Code and Revision

For this audit, we performed research, investigation, and review of the FermionToken contract followed by issue reporting, along with mitigation and remediation instructions outlined in this report. The following code files are considered in-scope for the review:

Source:

https://github.com/fermionlabs/fermion - commit [432a1c414f2cf55636b41ad5383a7a1264fb7136]

2.2 Attacks made to the contract

In order to check for the security of the contract, we tested several attacks in order to make sure that the contract is secure and follows best practices.

№	Issue description.	Checking status
1	Compiler warnings.	PASSED
2	Race conditions and Reentrancy. Cross-function race conditions.	PASSED
3	Possible delays in data delivery.	PASSED
4	Oracle calls.	PASSED
5	Front running.	PASSED
6	Timestamp dependence.	PASSED
7	Integer Overflow and Underflow.	PASSED
8	DoS with Revert.	PASSED
9	DoS with block gas limit.	PASSED
10	Methods execution permissions.	PASSED
11	Economy model. If application logic is based on an incorrect economic model, the application would not function correctly and participants would incur financial losses. This type of issue is most often found in bonus rewards systems, Staking and Farming contracts, Vault and Vesting contracts, etc.	PASSED
12	The impact of the exchange rate on the logic.	PASSED
13	Private user data leaks.	PASSED
14	Malicious Event log.	PASSED
15	Scoping and Declarations.	PASSED
16	Uninitialized storage pointers.	PASSED
17	Arithmetic accuracy.	PASSED

18	Design Logic.	MEDIUM ISSUES
19	Cross-function race conditions.	PASSED
20	Safe Zeppelin module.	PASSED
21	Fallback function security.	PASSED
22	Overpowered functions / Owner privileges	MEDIUM ISSUES

3.0 Security Issues

3.1 High severity issues [0]

No high severity issues found.

3.2 Medium severity issues [2]

1. Onchain random generation

When an investor buys Fermion tokens they have the chance to receive an additional % of tokens (calculated via oracle).

This % of tokens is calculated by a random generated onchain taking into account the following parameters:

Timestamp, number, hash and difficulty of the block.

Miners have the choice of whether to publish a block or not so there is a risk that someone manipulates the rewards of each purchase (choosing only the transactions that have the higher reward rate).

Recommendation:

Use external oracles such as VRF chainlink and not avoid showing the rewards at the mempool block (separate purchase from rewards).

2. Centralization:

Owner has authority over the next functions:

LiquidityLock contract (already locked for 1 year):

setOwner(): Owner can change the address to a new owner address.

acceptOwnership(): Owner can make a 'pendingowner' that will make the new owner need to accept the role.

acceptOwnership(): Execute the 'pendingowner' requirement to receive the owner privilege.

ExtenLockTime(): Owner can extend the lock time (no limits).

Withdraw(): Owner can withdraw the locked tokens only if the lock is finished (you can check the end variable to see when it is).

<u>Oracle contract:</u>

addAdmin(): Owner (admin) can add new admins.

addReader(): Owner (admin) can add new roles.

addRand(): Owner (admin) can add new rands.

setRand() / feedRandomness() : Owner (admin) can set rands.

mixRandom(): Owner (admin) can mix rands.

<u>Rewards contract:</u>

setRewardSheetAddress(): Owner (admin) can set a different contract for the reward sheet.

setPriceDataAddress(): Owner (admin) can set a different contract for the price data (aggregator interface).

addAdmin(): Owner (admin) can add new admins.

addReader(): Owner (admin) can add new roles.

resetIncrementer(): Owner (admin) can reset incrementer to 0.

excludeFromFee(): Owner (admin) can exclude address from fees.

includeInFee(): Owner (admin) can include address to fees.

updateOracleAddress(): Owner (admin) can change the oracle address.

<u>Token contract:</u>

setGlobalTax(): Owner can set a global tax (no limits).

setTaxBuy(): Owner can set a tax for purchase orders (no limits).

setTaxSell(): Owner can set a tax for sell orders (no limits).

setPrice(): Owner can edit the sale price of the token.

switchPause(): Owner can pause the contract.

setMaxTxPercent(): Owner can set a maximum limit of % transactions.

setMaxTxAmount(): Owner can set max tx to be done (it can be 0).

setTaxFee(): Owner can set a tax fee (no limits).

foundationWithdraw(): Owner can withdraw the funds to the foundation wallet.

UpdateRewardAddress(): Owner can change the address that receives the fees.

updateFundsWallet(): Owner can change the funds wallet address.

burnTokens(): Owner can burn tokens.

<u>Recommendation:</u>

-Add a time lock on privileged operations.

-Use a multisig wallet to prevent SPOF on the Private Key.

-Introduce DAO mechanism for owner functions (will add transparency and user involvement).

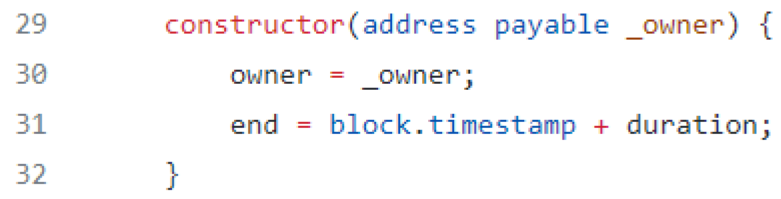
3.3 Low severity issues [1]

1. Owner not set at the creation of the contract.

There is a risk that the owner can be 0 address at the constructor of the contract.

Recommendation:

Change to [example]: setOwner(msg.sender);



```
    constructor(address payable _owner) {
        owner = _owner;
        end = block.timestamp + duration;
    }
```

4.0 Summary of the audit

Fermion token is an experimental protocol that is attempting to innovate bonus delivery using a new bonus mechanism. As with all innovations, new risks will likely be discovered in the future and will be outside the scope of this report. Additionally, the risks and vulnerabilities outlined in this report have not been remediated as of the publishing of this report. We encourage the team to monitor all transactions and pause smart contract activity should anomalous activity or targeted exploitation occur.