TCG 2

ERC20

AUDITED BY

Disclaimer

Accepting a project audit can be viewed as a sign of confidence and is typically the first indicator of trust for a project, but it does not guarantee that a team will not remove all liquidity, sell tokens, or engage in any other type of fraud. There is also no method to restrict private sale holders from selling their tokens. It is ultimately your obligation to read through all documentation, social media posts, and contract code for each particular project in order to draw your own conclusions and define your own risk tolerance.

DRIVENsecurity accepts no responsibility for any losses or encourages speculative investments. This audit's material is given solely for informational reasons and should not be construed as investment advice.

Request an audit for your project at

office@drivenx.finance

DRIVEN
SECURITY

Project Details

Project name: TCG World

Project type: ERC20 Token

Contract Address:
ETH Address:
0x0d31DF7dedd78649A14aAe62D99CcB23aBCC3A5A

Polygon Address:
0x064424bE39D59EfBE2601027a6338D03DDE3d476

Blockchain: Ethereum & Polygon

Token Name:
ETH: TCG 2.0 ETH
Polygon: TCG 2.0 Polygon

Token Ticker:
ETH: ethTCG2
Polygon: polyTCG2

Decimals: 9

Project Website: www.tcg.world

This audit was created on: 20 Dec. 2021

Overview

General issues

- Security issues: [passed]
- Gas & Fees issues: [passed]
- ERC errors: [passed]
- Compilation errors: [passed]
- Design logic: [passed]
- Timestamp dependence: [passed]
- Buy & sell: [passed]

Security against cyber-attacks

- Private user's data: [secured]
- Reentrancy: [secured]
- Cross-function Reentrancy: [passed]
- Front Running: [passed]
- Taxonomy attacks: [passed]
- Integer Overflow and Underflow: [passed]
- DoS (Denial of Service) with Unexpected revert: [passed]
- DoS (Denial of Service) with Block Gas Limit: [passed]
- Insufficient gas griefing: [passed]
- Forcibly Sending BNB to a Contract: [passed]

In-depth analysis

Functions that can be called by owner

Exclude / include address from fees or/and rewards or blacklist/whitelist:

- *excludeFromFee* - exclude address from paying fees;
- *includeInFee* - the account will pay fees;
- *setupBot* -blacklist a malicious address;

Fees and max amounts:

- *setLiquidityFeePercent* - set the liquidity fee;
- *setLiquiditySellFeePercent* - set the liquidity fee on sells;
- *setMaxTxPercent* - set the max percent of the supply that can be bought/sold/transfered.

Fees for DEX listings:

- *setDexWitBuyFee, setDexWithSellFee* - allow buy/sell fees on DEXs;

In-depth analysis

- *setNumTokensSellToAddToLiquidity* - set the amount of tokens that will be liquidated by the smart contract to create liquidity tokens;
- *setTaxFeeFlag;*
- *setTaxFeePercent;*

Other functions:

- *rescueTokensFromContract* - withdraw ETH/TCG tokens from the contract if somebody sent them by accident;
- *setMaxLoopCount* - avoid infinity loops;
- *setSwapAndLiqufityEnabled* - enable swap and liquifity tokens;
- *swapAndLiquifyByOwner* - manually liquify tokens;
- *transferOwnership;*
- *lock & unlock;*

Anti-Bot Function

TCG Wold team included an anti-bot mechanism that can be triggered by the smart contract owner to protect investor funds and prevent bot activity on the TCG smart contract.

How does this function work?

The _isBot[account] variable is set to true or false by the "setupBot" function. This variable is used in the TCG token smart contract's to enable/disable address from buying, selling, and transferring tokens.

If TCG World's team detects a bot action or one of their investors reports a compromised wallet, the contract owner can execute a function with the value "True" and then apply the compromised wallet or bot's address. After then, unless the "setupBot" function value is turned back to "False," that address will be unable to transact any TCG tokens.

Approved by DRIVENsecurity team

Penetration testing

Re-entrancy

What is "Re-entrancy"?

A re-entrancy attack can arise when you write a function that calls another untrusted contract before resolving any consequences. If the attacker has authority over the untrusted contract, he can initiate a recursive call back to the original function, repeating interactions that would otherwise not have occurred after the effects were resolved.

Attackers can take over the smart contract's control flow and make modifications to the data that the calling function was not anticipating.

To avoid this, make sure that you do not call an external function until the contract has completed all of the internal work.

Test: [passed]

Penetration testing

Cross-function Re-entrancy

What is "Cross-function Reentrancy"?

When a vulnerable function shares the state with another function that has a beneficial effect on the attacker, this cross-function re-entrancy attack is achievable. This re-entrancy issue that is the employment of intermediate functions to trigger the fallback function and a re-entrancy attack is not unusual.

Attackers can gain control of a smart contract by calling public functions that use the same state/variables as "private" or "onlyOwner" functions.

To avoid this, make sure there are no public functions that use private variables, and avoid calling routines that call external functions or use mutex (mutual exclusion).

Test: [passed]

Penetration testing

Front Running

What is "Front Running"?

Front-running indicates that someone can obtain prior information of transactions from other beneficial owners by technology or market advantage, allowing them to influence the price ahead of time and result in economic benefit, which usually results in loss or expense to others.

Since all transactions are visible in the block explorer for a short period of time before they are executed, network observers can see and react to an action before it is included in a block.

Attackers can front-run transactions because every transaction is visible to the blockchain, even if it is in the "processing" or "indexing" state. This is a very low security vulnerability because it is based on the blockchain rather than the contract.

The only possible attack is seeing transactions made by bots. Using transaction fees, you can avoid bots.

Test: [passed]

Penetration testing

Taxonomy attacks

Those taxation attacks can be made in 3 ways:

1) Displacement - performed by increasing the gasPrice higher than network average, often by a multiplier of 10.

2) Insertion - outbidding transction in the gas price auction.

3) Suppression (Block Stuffing) - The attacker sent multiple transactions with a high gasPrice and gasLimit to custom smart contracts that assert to consume all the gas and fill up the block's gasLimit.

This type of attacks occurs mainly for exchanges, so this smart contract is secured.

Test: [passed]

Penetration testing

Integer Overflow and Underflow

- overflow: An overflow occurs when a number gets incremented above its maximum value.
In the audited contract: uint8 private _decimals = 9;

Test: [passed]

(decimals can't reach a value bigger than it's limit)

- underflow: An overflow occurs when a number gets decremented below its maximum value.

Test: [passed]

(there are no decrementation functions for parameters and users can't call functions that are using uint values);

This contract use the update version of SafeMath for uint, int and mathematical operations.

Test: [passed]

Penetration testing

DoS (Denial of Service) with Unexpected revert

DoS (Denial of Service) attacks can occur in functions when you attempt to transmit funds to a user and the functionality is dependent on the successful transfer of funds.

This can be troublesome if the funds are given to a bad actor's smart contract (when they call functions like “Redeem” or “Claim”), since they can simply write a fallback function that reverts all payments.

Test: [passed]

There are no functions that deliver money to users, attackers are unable to communicate using a contract with fallBack functions

Penetration testing

DoS (Denial of Service) with Block Gas Limit

Each block has an upper bound on the amount of gas that can be spent, and thus the amount computation that can be done. This is the Block Gas Limit. If the gas spent exceeds this limit, the transaction will fail. This leads to a couple possible Denial of Service vectors.

Test: [passed]

Penetration testing

Insufficient gas griefing

This attack can be carried out against contracts that accept data and use it in a sub-call on another contract.

This approach is frequently employed in multisignature wallets and transaction relayers. If the sub-call fails, either the entire transaction is rolled back or execution is resumed.

Test: [passed]

Users can't execute sub-calls.

Penetration testing

Forcibly Sending ETH to a Contract

Test: [passed]

Notes

This Smart Contract was tested on Rinkeby Testnet by both TCG World Team and DRIVENsecurity team.

TCG World Test: https://ropsten.etherscan.io/token/0xd8cbC1F8370BeCC1D9B1c83D87eE1F66C68c6B61

DRIVENsecurity Test: https://rinkeby.etherscan.io/address/0xc26865a598980229cbd66e5e7c3ec3d633e26d7a

Thank you!

Request an audit at
office@drivenx.finance

DRIVEN
SECURITY