eDOT

AUDITED BY

DRIVEN security

Disclaimer

Accepting a project audit is a sign of confidence and is usually 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 way to prevent private sale token holders from selling their tokens. It is ultimately your responsibility to read all documentation, social media posts, and contract code for each particular project in order to draw your own conclusion and define your own risk tolerance.

DRIVENsecurity accepts no responsibility for any losses or 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: eDot

Contract Address:
0xa99c3aDe9b6D4c9a1b786Afae64AD62ab194E0bf

Contract Creator:
0xe32C3F7Fb9Aba693Fc92E5C638874ba782E26474

Blockchain: Binance Smart Chain / Solidity

Contract Name: EarnDOT

Token Ticker: eDOT

Decimals: 18

Transactions Count: 6,811

Project Website: https://www.earndot.net/

First check: 30 Sept. 2021

Overview (2nd check)

General issues

- Security issues 1 low-severity issue
- Gas & Fees issues: 1 low-severity issue
- ERC errors: [passed]
- Compilation errors: [passed]
- Design logic: [passed]
- Timestamp dependence: [passed]

Security against cyber-attacks

- Private user's data: [secured]
- Reentrancy: 1 low-severity issue
- 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

Gas and Fees issues: 1 low-severity issue

We observed that the following functions:

- excludeMultipleAccountsFromFees
- process

are using FOR loops with dynamic arrays, so the number of iterations is uncontrolled.

In-depth analysis

Security issues: 1 low-severity issue

We observed that the contract is using “tx.origin”:

- emit ProcessedDividendTracker(iterations, claims, lastProcessedIndex, false, gas, tx.origin);
- emit ProcessedDividendTracker(iterations, claims, lastProcessedIndex, true, gas, tx.origin);

The contract is secured because those are only emitted events and the contract does not use “tx.origin” for authorization purposes.

Functions:

- _withdrawDividendOfUser(address payable user) internal returns (uint256) [DividendPayingToken.sol]
- _transfer

are vulnerable for Re-entrancy attacks.

In-depth analysis

Compilation issues: 0

Functions that can be called by owner

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

- blacklistAddress
- excludeFromDividends
- excludeFromFees
- excludeMultipleAccountsFromFees

Ownership:

- renounceOwnership;
- transerOwnership - transfer the ownership to another adddress;

In-depth analysis

Functions that can be called by owner

Set fees & gass:

- setLiquidityFee
- setMarketingFee
- setTaxFeeFlag - indicates if fee should be deducted from transfer ;
- setPolkadotRewardsFee
- updateGasForProcessing - update the gas amount which the contract will use to process dividends.

Other functions:

- updateDividendTracker - change the address of DividendTracker;
- updateClaimWait - set the time when holders receive dividends;
- setMaxTxPercent - set the maximum value of transactions;
- setMarketingWallet – change the address of the marketing wallet

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: low-severity issue

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 BNB to a Contract

Test: [passed]

Thank you!

Request an audit at
office@drivenx.finance

DRIVEN.
security