



Elden Doge

Smart Contract Audit Report



ABOUT AUDITACE

Audit Ace is built, to combat financial fraud in the cryptocurrency industry, a growing security firm that provides audits, Smart contract creation, and end-to-end solutions to all crypto-related queries.

Website - <https://auditace.tech/>

Telegram - https://t.me/Audit_Ace

Twitter - https://twitter.com/auditace_

Github - <https://github.com/Audit-Ace>



Overview

AUDITACE team has performed a line-by-line manual analysis and automated review of smart contracts. Smart contracts were analyzed mainly for common contract vulnerabilities, exploits, and manipulation hacks.

Audit Result: Passed - **Low Risk**

Audit Date: November 23, 2022

KYC:Done

Audit Team: TEAM AUDITACE

Result details: no centralization or logical issues found in the contract, token launched on a local blockchain and all functionalities were tested.



Disclaimer

All the content provided in this document is for general information only and should not be used as financial advice or a reason to buy any investment. Team provides no guarantees against the sale of team tokens or the removal of liquidity by the project audited in this document. Always Do your own research and protect yourselves from being scammed. The Auditace team has audited this project for general information and only expresses their opinion based on similar projects and checks from popular diagnostic tools. Under no circumstances did Auditace receive a payment to manipulate those results or change the awarding badge that we will be adding in our website. Always Do your own research and protect yourselves from scams. This document should not be presented as a reason to buy or not buy any particular token. The Auditace team disclaims any liability for the resulting losses.

Elden Doge

Social Media Overview



<https://t.me/EldenDogeToken>



<https://www.eldendoge.com/>



<https://twitter.com/EldenDoge>



Token Summary

Parameter	Result
Address	0x8516E3e58b4b9aF2A66C14E7853236FedFD05809
Token Type	BEP 20
Decimals	9
Supply	1,000,000,000
Platform	Binance Smart Chain
Compiler	v0.8.17+commit.8df45f5f
Contract Checksum	2bfe0ce62c654dd3c96f2982faf536b8bb33b4053145a3ea1ab 5913ed5e381db

Used Tools

Manual Review - Forked Pancakeswap V2 on local blockchain

Tests:

- adding liquidity with WBNB and BUSD
- buying and selling right after launch
- taxes get collected inside the contract
- taxes are sent marketing wallet (BNB) and autoliquidity works

The screenshot shows a Visual Studio Code editor window titled 'HasWBNB.js - backend contact checker - Visual Studio Code'. The Explorer sidebar on the left displays the project structure for 'Vitalik-Platform-Contract-Checker-backend.zip', with 'honeycheck.js' selected under the 'honeycheckers' directory. The main editor area shows the content of 'honeycheck.js', which is a JavaScript file using ethers.js to interact with a DEX. The code defines a function 'buyGas' that takes a token, signer, signerAddress, Dex, and ethAddress as arguments. It calculates the gas cost for a transaction and sends it to the DEX.

```

1  const { ethers, network } = require("hardhat");
2
3  const HasWBNB = async (token, signer, signerAddress, Dex, ethAddress) => {
4    let buyGas;
5    let sellGas;
6    let tokenName = await token.symbol();
7    const toGet = {
8      await Dex.getAmountsOut(ethers.utils.parseEther("1"), [
9        ethAddress,
10       token.address,
11     ])
12   };
13   console.log("Expect To Get " + toGet);
14   const pn_buy_tx =
15     await Dex.swapExactETHforTokensSupportingFeeOnTransferTokens(
16       0,
17       [ethAddress, token.address],
18       signerAddress,
19       Date.now() + 1000 * 60,
20       {
21         value: ethers.utils.parseEther("1"),
22         gasPrice: await ethers.provider.getGasPrice(),
23         gasLimit: "1000000",
24       }
25     );
26   const buy_rec = await pn_buy_tx.wait(1);
27   buyGas = buy_rec.gasUsed;
28   console.log("• Bought...");
29
30   let added = (await token.functions.balanceOf(signerAddress)).balance;
31   let buyTax = toGet
32     .sub(added)
33     .mul(10 ** 4)
34     .div(toGet);
35   buyTax = parseEther(buyTax) / 100;
36   console.log("Buy Tax : " + buyTax);
37   await network.provider.send("evm_increaseTime", [5]);

```

NOT DETECTED

NOT DETECTED

NOT DETECTED

NOT DETECTED

NOT DETECTED

NOT DETECTED

NOT DETECTED

AUDIT METHODOLOGY

The auditing process will follow a routine as special considerations by Auditace:

- Review of the specifications, sources, and instructions provided to Auditace to make sure the contract logic meets the intentions of the client without exposing the user's funds to risk.
 - Manual review of the entire codebase by our experts, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
 - Specification comparison is the process of checking whether the code does what the specifications, sources, and instructions provided to Auditace describe.
 - Test coverage analysis determines whether the test cases are covering the code and how much code is exercised when we run the test cases.
 - Symbolic execution is analysing a program to determine what inputs cause each part of a program to execute.
 - Reviewing the codebase to improve maintainability, security, and control based on the established industry and academic practices.
-



Issues Checking Status

No	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	Design Logic.	Passed
12	Cross-function race conditions.	Passed
13	Safe Zeppelin module.	Passed
14	Malicious Event log.	Passed
15	Scoping and Declarations.	Passed
16	Fallback function security.	Passed
17	Arithmetic accuracy.	Passed



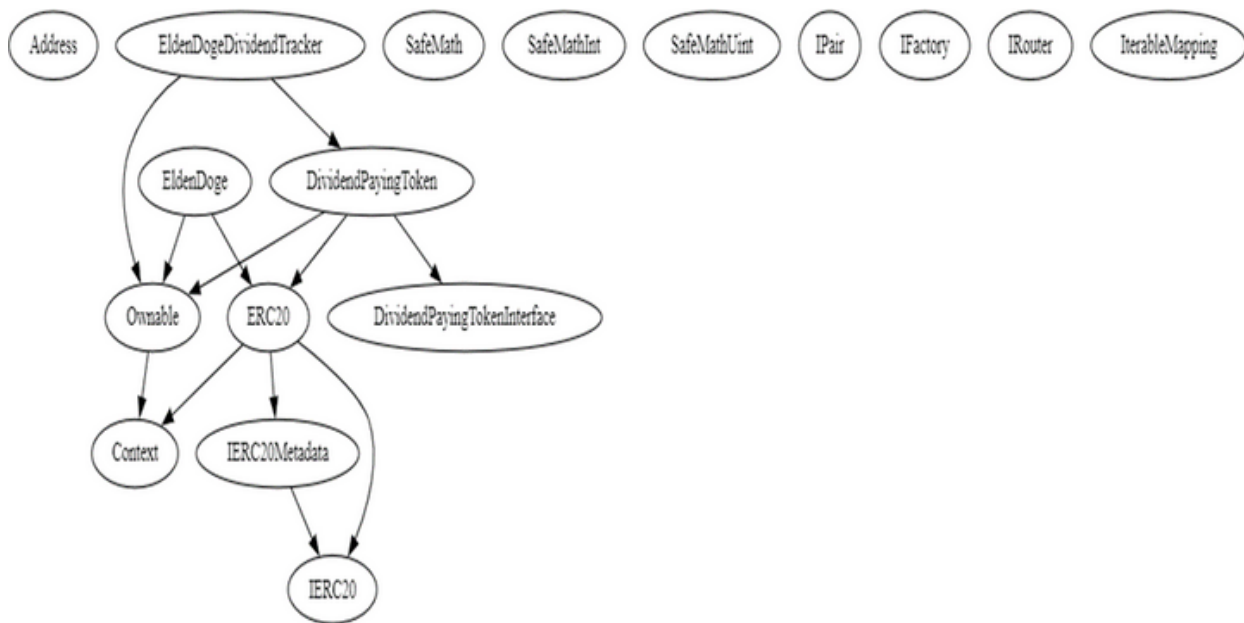
SWC ATTACK TEST

SWC ID	Description	Test Result
SWC-100	Function Visibility	Passed
SWC-101	Integer Overflow and Underflow	Passed
SWC-102	Outdated Compiler Version	Passed
SWC-103	Floating Pragma	Passed
SWC-104	Unchecked Call Return Value	Passed
SWC-105	Unprotected Ether Withdrawal	Passed
SWC-106	Unprotected SELFDESTRUCT Instruction	Passed
SWC-107	Re-entrancy	Passed
SWC-108	State Variable Default Visibility	Passed
SWC-109	Uninitialized Storage Pointer	Passed
SWC-110	Assert Violation	Passed
SWC-111	Use of Deprecated Solidity Functions	Passed
SWC-112	Delegate Call to Untrusted Callee	Passed
SWC-113	DoS with Failed Call	Passed
SWC-114	Transaction Order Dependence	Passed
SWC-115	Authorization through tx.origin	Passed
SWC-116	Block values as a proxy for time	Passed



SWC ID	Description	Test Result
SWC-117	Signature Malleability	Passed
SWC-118	Incorrect Constructor Name	Passed
SWC-119	Shadowing State Variables	Passed
SWC-120	Weak Sources of Randomness from Chain Attributes	Passed
SWC-121	Missing Protection against Signature Replay Attacks	Passed
SWC-122	Lack of Proper Signature Verification	Passed
SWC-123	Requirement Violation	Passed
SWC-124	Write to Arbitrary Storage Location	Passed
SWC-125	Incorrect Inheritance Order	Passed
SWC-126	Insufficient Gas Grieving	Passed
SWC-127	Arbitrary Jump with Function Type Variable	Passed
SWC-128	DoS With Block Gas Limit	Passed
SWC-129	Typographical Error	Passed
SWC-130	Right-To-Left-Override control character (U+202E)	Passed
SWC-131	Presence of unused variables	Passed
SWC-132	Unexpected Ether balance	Passed
SWC-133	Hash Collisions with Multiple Variable Length Arguments	Passed
SWC-134	Unencrypted Private Data On-Chain	Passed

Inheritance Tree



Summary

- Anti-Bot implementation : for up to 5 blocks, buyers and sellers get taxes by 99%
 - Owner is not able to change taxes,(3% buy and 3% sell static)
 - Owner is not able to set max buy/sell/transferring amount
 - Owner is not able to mint new tokens
 - Owner is not able to pause trades
 - Owner must enable trading in order for investors to be able to trade
 - Owner is not able to blacklist an arbitrary address
-

Classification of Risks

Severity

Description

◆ High-Risk	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.
◆ Medium-Risk	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.
◆ Low-Risk	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.
◆ Gas Optimization / Suggestion	A vulnerability that has an informational character but is not affecting any of the code.

Findings

Severity

Found

◆ High-Risk	0
◆ Medium-Risk	2
◆ Low-Risk	0
◆ Gas Optimization / Suggestions	3



MANUAL AUDIT

Medium Risk Findings

Centralization - owner is able to update dividend tracker, changing dividend tracker to a malicious contract may effect trades (eg disable them)

logical - setting swapTokensAtAmount to 0 can disable sells if collected tokens from taxes (i.e contract's eldendoge token balance) is greater than swapTokensAtAmount.

Suggestions: make sure that you can not set swapTokensAtAmount to 0 at setSwapTokensAtAmount

MANUAL AUDIT

Gas Optimizations

- do not use SafeMath library, overflow/underflows are handled internally by compiler if compiler's version is more than 0.8.0, using safemath only increases gas usage.
- create 2 variables for total buy taxes and total sell taxes, this will decrease gas usage in _transfer function which leads to lower gas usage for all buys/sells/transfers, currently SLOAD opcode costs 2100 gas

Suggestions

- do not use SafeMath library, overflow/underflows are handled internally by compiler if compiler's version is more than 0.8.0
-