Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Customer: Gem Hunters
Website: https://wsjackpot.com
Platform: Binance Smart Chain
Language: Solidity
Date: September 2nd, 2021

Table of contents

Email: audit@EtherAuthority.io

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO THE PUBLIC AFTER ISSUES ARE RESOLVED.

Email: audit@EtherAuthority.io

Introduction

EtherAuthority was contracted by the WallStreetJackpot team to perform the Security audit of the WallStreetJackpot Token smart contract code. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on September 2nd, 2021.

The purpose of this audit was to address the following:

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

Project Background

WallStreet JACKPOT is a token created by a team with strong fundamental knowledge on the Binance Smart Chain. WallStreet JACKPOT is a frictionless dividend reward generating token that automatically distributes BNB to holders every 60 minutes.
The token is based on a dividend-paying token standard, which means all BNB contract gains will be split equally proportionally between the token holders.

Audit scope

Name	**Code Review and Security Analysis Report for WallStreetJackpot Token Smart Contract**
Platform	**BSC / Solidity**
File	WallStreetJackpot.sol
File MD5 Hash	9EE760DC5FEAFEAA8430D31F73C50E4C
Audit Date	September 2nd, 2021

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Name: WallStreet Jackpot • Symbol: WSJ • Decimals: 18 • Total Supply: 100 Billion • Platform: BSC • BNB Rewards Fee: 6% • Jackpot Pool Fee: 6% • Liquidity Fee: 2% • Marketing Fee: 4% • Total Fees: 18% • Developer Wallet Fee: 3% • Contest Reward Fee: 5% • DXSale Fee: 10% • Liquidity: 34% • Presale: 48%	**YES, This is valid.**
100 Billion tokens will be minted and sent to a marketing wallet address at the time of contract deployment. New tokens can never be minted	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity smart contracts are **“Secured”**. This token contract does not contain any owner control, making it fully decentralized.

We used various tools like Slither, Solhint and Remix IDE. At the same time this finding is based on critical analysis of the manual audit.
All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit overview section. General overview is presented in AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium and 3 low and some very low level issues.

Investors Advice: Technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Solidity version too old	Moderated
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Moderated
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Moderated
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	Passed
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Passed
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Moderated
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Email: audit@EtherAuthority.io

Code Quality

This audit scope has 1 smart contract file. Smart contracts also contain Libraries, Smart contracts, inherits and Interfaces. These are compact and well written contracts.

The libraries in WallStreetJackpot are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties / methods can be reused many times by other contracts in the Wall Street Jackpot token.

The WallStreetJackpot Token team has **not** provided scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are **not well** commented on smart contracts.

Documentation

We were given a WallStreetJackpot smart contracts code in the form of a BscScan web link. The hashes of that code are mentioned above in the table.

As mentioned above, code parts are **not well** commented. So it is not easy to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Another source of information was its official website https://wsjackpot.com/ which provided rich information about the project architecture and tokenomics.

Use of Dependencies

As per our observation, the libraries are used in this smart contract infrastructure that are based on well known industry standard open source projects. And their core code blocks are written well.

Apart from libraries, its functions are not used in external smart contract calls.

AS-IS overview

(1) Interface

(a) IUniswapV2Factory

(b) IUniswapV2Pair

(c) IUniswapV2Router01

(d) IUniswapV2Router02

(e) IERC20

(f) IERC20Metadata

(g) DividendPayingTokenOptionalInterface

(h) DividendPayingTokenInterface

(2) Inherited contracts

(a) ERC20

(b) Context

(c) Ownable

(d) Jackpot

(e) DividendPayingToken

(f) WallStreetJackpot

(3) Usages

(a) using SafeMath for uint256;

(4) Events

(a) event UpdateDividendTracker(address indexed newAddress,address indexed oldAddress);

(b) event UpdateJackpot(address indexed newAddress, address indexed oldAddress);

(c) event UpdateUniswapV2Router(address indexed newAddress,address indexed oldAddress);

(d) event ExcludeFromFees(address indexed account, bool isExcluded);

(e) event ExcludeMultipleAccountsFromFees(address[] accounts, bool isExcluded);

(f) event SetAutomatedMarketMakerPair(address indexed pair, bool indexed value);

(g) event LiquidityWalletUpdated(address indexed newLiquidityWallet,address indexed oldLiquidityWallet);

(h) event GasForProcessingUpdated(uint256 indexed newValue,uint256 indexed oldValue);

(i) event SwapAndLiquify(uint256 tokensSwapped, uint256 ethReceived, uint256 tokensIntoLiqudity);

(j) event SendDividends(uint256 tokensSwapped, uint256 amount);

(k) event ProcessedDividendTracker(uint256 iterations, uint256 claims, uint256 lastProcessedIndex, bool indexed automatic, uint256 gas, address indexed processor);

(5) Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	_transfer	internal	Passed	No Issue
3	owner	read	Passed	No Issue
4	onlyOwner	modifier	Passed	No Issue
5	renounceOwnership	write	access only Owner	No Issue
6	transferOwnership	write	access only Owner	No Issue
7	receive	external	Passed	No Issue
8	distributeDividends	write	Passed	No Issue
9	withdrawDividend	write	Passed	No Issue
10	_withdrawDividendOfUser	internal	Passed	No Issue
11	dividendOf	read	Passed	No Issue
12	withdrawableDividendOf	read	Passed	No Issue
13	withdrawnDividendOf	read	Passed	No Issue
14	accumulativeDividendOf	read	Passed	No Issue
15	_transfer	internal	Division before multiplication	Refer audit findings section below
16	_mint	internal	Passed	No Issue
17	_burn	internal	Passed	No Issue
18	_setBalance	internal	Passed	No Issue
19	owner	read	Passed	No Issue
20	onlyOwner	modifier	Passed	No Issue
21	renounceOwnership	write	access only Owner	No Issue
22	transferOwnership	write	access only Owner	No Issue
23	addLiquidity	write	Centralized risk in addLiquidity	Refer audit findings section below
24	receive	external	Passed	No Issue

25	setPrize	external	access only Owner	No Issue
26	setPrizePercent	external	access only Owner	No Issue
27	setPrizeOdds	external	access only Owner	No Issue
28	switchJackpot	write	access only Owner	No Issue
29	setJackpotPeriod	write	access only Owner	No Issue
30	setJackpotThreshold	write	access only Owner	No Issue
31	setTokensAmont	write	Function input parameters lack of check	Refer audit findings section below
32	enableBasedOnToken	write	access only Owner	No Issue
33	setEnableSubTickets	write	access only Owner	No Issue
34	excludeJackpot	write	Function input parameters lack of check	Refer audit findings section below
35	setDxsale	write	Function input parameters lack of check	Refer audit findings section below
36	setPresalePrice	write	Function input parameters lack of check	Refer audit findings section below
37	updateJackpot	write	access only Owner	No Issue
38	updateDividendTracker	write	access only Owner	No Issue
39	updateUniswapV2Router	write	Critical operation lacks event log	Refer audit findings section below
40	excludeFromFees	write	access only Owner	No Issue
41	excludeMultipleAccountsFromFees	write	Infinite loop	Refer audit findings section below
42	setMarketWallet	external	Critical operation lacks event log	Refer audit findings section below
43	setBNBRewardsFee	external	Function input parameters lack of check	Refer audit findings section below
44	setLiquiditFee	external	Function input parameters lack of check	Refer audit findings section below

45	setMarketingFee	external	Function input parameters lack of check	Refer audit findings section below
46	setJackpotFee	external	Function input parameters lack of check	Refer audit findings section below
47	setAutomatedMarketMaker Pair	write	access only Owner	No Issue
48	blacklistAddress	external	access only Owner	No Issue
49	_setAutomatedMarketMaker Pair	write	Passed	No Issue
50	updateGasForProcessing	write	access only Owner	No Issue
51	updateClaimWait	external	access only Owner	No Issue
52	getClaimWait	external	Passed	No Issue
53	getTotalDividendsDistributed	external	Passed	No Issue
54	isExcludedFromFees	read	Passed	No Issue
55	withdrawableDividendOf	read	Passed	No Issue
56	dividendTokenBalanceOf	read	Passed	No Issue
57	excludeFromDividends	external	Missing Error Message	Refer audit findings section below
58	getAccountDividendsInfo	external	Passed	No Issue
59	getAccountDividendsInfoAtIndex	external	Passed	No Issue
60	processDividendTracker	external	Passed	No Issue
61	claim	external	Passed	No Issue
62	getLastProcessedIndex	external	Passed	No Issue
63	getNumberOfDividendToken Holders	external	Passed	No Issue
64	_transfer	internal	Passed	No Issue
65	setDxSaleAddress	external	Critical operation lacks event log	Refer audit findings section below
66	enableTrading	write	access only Owner	No Issue
67	swapAndSendToFee	write	Critical operation lacks event log	Refer audit findings section below
68	swapAndSendToJackpot	write	Critical operation lacks event log	Refer audit findings section below
69	swapAndLiquify	write	Function input parameters lack of check	Refer audit findings section below

70	swapTokensForEth	write	Function input parameters lack of check	Refer audit findings section below
71	swapAndSendDividends	write	Passed	No Issue

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored.

Email: audit@EtherAuthority.io

Audit Findings

Critical

No Critical severity vulnerabilities were found.

High

No High severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

(1) Centralized risk in addLiquidity:

```
function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {
    // approve token transfer to cover all possible scenarios
    _approve(address(this), address(uniswapV2Router), tokenAmount);

    // add the liquidity
    uniswapV2Router.addLiquidityETH{value: ethAmount}(
        address(this),
        tokenAmount,
        0, // slippage is unavoidable
        0, // slippage is unavoidable
        _marketingWalletAddress,
        block.timestamp
    );
}
```

In addLiquidity function, _marketingWalletAddress gets WSJ Tokens from the Pool. If the private key of the _marketingWalletAddress would be compromised, then it will create a problem.

Resolution: Ideally this can be a governance smart contract. On the other hand, _marketingWalletAddress can accept this risk and handle the private key very securely.
Status: Open.

(2) Infinite loop:
The excludeMultipleAccountsFromFees function for loop does not have accounts length limit , which costs more gas.

Resolution: Upper limit should be limited in for loops.

Status: Open.

(3) Function input parameters lack of check:

Variable validation is not performed in below functions : swapAndLiquify = tokens | setJackpotThreshold = jackpotThreshold | setTokensAmont = tokensAmont | excludeJackpot = holder | setPresalePrice = price | setDxsale = dxRouter , presaleRouter | setJackpotFee = value | setBNBRewardsFee = value | setLiquiditFee = valuie | setMarketingFee = value |

Resolution: put validation : variable is not empty & > 0 and for address type check variable is not address(0). To set the fee as a percentage , please validate not greater than 100.

Status: Open.

Very Low / Informational / Best practices:

(1) Use the latest solidity version:

```
pragma solidity ^0.6.2;
```

Using the latest solidity will prevent any compiler-level bugs.

Resolution: Please use 0.8.7 which is the latest version.

Status: Open.

(2) Critical operation lacks event log:

Missing event log for : swapAndSendToJackpot , swapAndSendToFee , setDxSaleAddress , setMarketWallet , updateUniswapV2Router.

Resolution: Please write an event log for listed events.

Status: Open.

(3) Missing Error Message:

```
function excludeFromDividends(address account) external onlyOwner {
    require(!excludedFromDividends[account]);
    excludedFromDividends[account] = true;

    _setBalance(account, 0);
    tokenHoldersMap.remove(account);

    emit ExcludeFromDividends(account);
}
```

Requirements must have error messages.

Resolution: Please write an error message.

Status: Open.

(4) Unused Variables:

_jackpoting variable is unused.

Resolution: Please remove unused variables.

Status: Open.

(5) Make the variable constant:

```
uint256 public _snipeBlockAmount = 3;
```

```
bool public _sniperProtection = true;
```

These variables remain unchanged.

Resolution: Declare those variables as constant. Just put the constant keyword.It will save some gas.

Status: Open.

(6) Confirm while deploy

```
address public _marketingWalletAddress =
    0xCa09441f094A9BdC553Aed501e1Be1145D88F72b;
```

_marketingWalletAddress is a hard-coded address set while deploying.

Resolution: Owner/Deployer has to double confirm before deploy, as this address got the minted tokens while deploying and also cannot be changed after deploy.

Status: Open.

(7) Division before multiplication

```
// jackpot
if ( jackpotEnabled) {
    jackpot.jackpot(uniswapV2Pair, from, to, amount);
}

bool takeFee = !swapping;
```

In this line, Jackpot.sol file's jackpot function has been called. In that function division has been done before multiplication at line 178.

Resolution: Solidity being resource constraint language, dividing any amount and then multiplying will cause discrepancies in the outcome. Therefore, always multiply the amount first and then divide it.

Status: Open.

Centralization

These smart contracts have some functions which can be executed by the Admin (Owner) only. If the admin wallet private key would be compromised, then it would create trouble. Following are Admin functions:

- setPrize: The Owner can set a jackpot prize.
- setPrizePercent: The Owner can set jackpot prize percentage.
- setPrizeOdds: The Owner can set jackpot prize odds.
- switchJackpot: The Owner can set the jackpot flag.
- setJackpotPeriod: The Owner can set the jackpot period.
- setJackpotThreshold: The Owner can set Jackpot Threshold.
- setTokensAmont: The Owner can set Tokens Amount.
- enableBasedOnToken: The Owner can set enable Based Token.
- setEnableSubTickets: The Owner can set Enable SubTickets.
- excludeJackpot: The Owner can exclude Jackpot holders.
- setDxsale: The Owner can set Dxsale.
- setPresalePrice: The Owner can set Presale Price.
- updateJackpot: The Owner can update the jackpot.
- updateDividendTracker: The Owner can update the dividend tracker.
- updateUniswapV2Router: The Owner can update uniswapv2router,
- excludeFromFees: The Owner can access exclude from fees.

- excludeMultipleAccountsFromFees: The Owner can exclude Multiple accounts from fees.
- setBNBRewardsFee: The owner can set a BNB rewards fee.
- setLiquiditFee: The Owner can set a liquidity fee.
- setMarketingFee: The Owner can set marketing fee.
- setJackpotFee: The Owner can set a jackpot fee.
- setAutomatedMarketMakerPair: The owner can set Automated Market Maker Pair.
- blacklistAddress: The Owner can set the blacklist address.
- updateGasForProcessing: The Owner can update gas for processing.
- updateClaimWait: The Owner can update the claim wait.
- excludeFromDividends: The Owner can access exclude from dividends.
- setDxSaleAddress: The Owner can set Dxsale address.
- enableTrading: The Owner can set enable trading, enable jackpot.

Conclusion

We were given a contract code. And we have used all possible tests based on given objects as files. We observed some issues in the smart contracts and those issues are not critical ones. So, **it's good to go to production**.

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

Audit report contains all found security vulnerabilities and other issues in the reviewed code.

Security state of the reviewed contract, based on standard audit procedure scope, is **"Secured".**

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

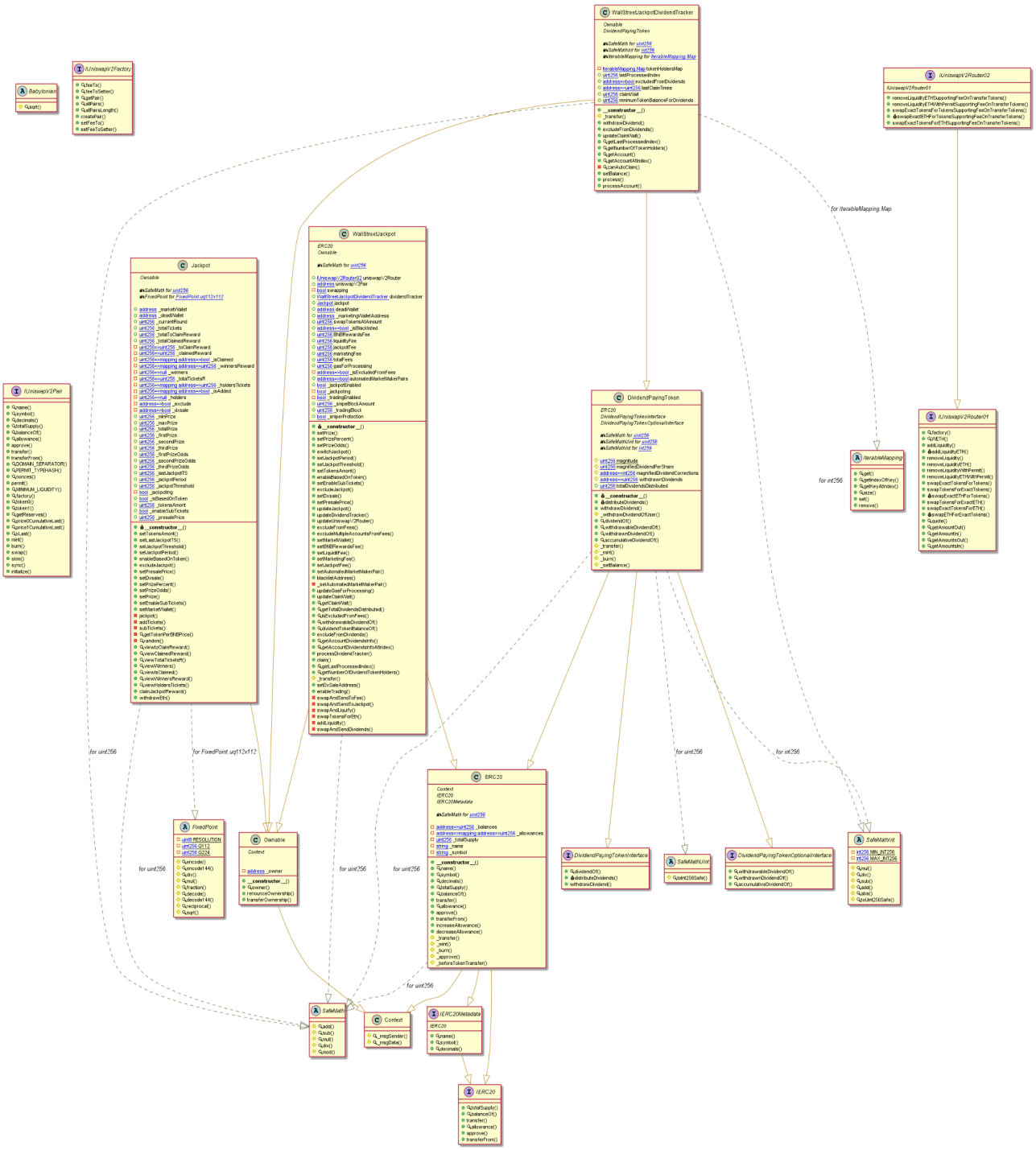
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can’t guarantee explicit security of the audited smart contracts.

Appendix

Code Flow Diagram - WallStreetJackpot Token



Slither Results Log

Slither log >> WallStreetJackpot.sol

```
INFO:Detectors:
WallStreetJackpot.addLiquidity(uint256,uint256) (WallStreetJackpot.sol#2596-2609) sends eth to arbitrary user
        Dangerous calls:
        - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,_marketingWalletAddress,block.timestamp) (WallS
treetJackpot.sol#2601-2608)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations
INFO:Detectors:
Jackpot.random(uint256) (WallStreetJackpot.sol#1501-1506) uses a weak PRNG: "uint256(keccak256(bytes)(abi.encodePacked(now,block.difficul
ty,msg.sender))) % _length (WallStreetJackpot.sol#1502-1505)"
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#weak-PRNG
INFO:Detectors:
Reentrancy in WallStreetJackpot._transfer(address,address,uint256) (WallStreetJackpot.sol#2398-2517):
        External calls:
        - swapAndSendToFee(marketingTokens) (WallStreetJackpot.sol#2456)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (W
allStreetJackpot.sol#2587-2593)
        - swapAndSendToJackpot(jackpotTokens) (WallStreetJackpot.sol#2461)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (W
allStreetJackpot.sol#2587-2593)
        - swapAndLiquify(swapTokens) (WallStreetJackpot.sol#2466)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,_marketingWalletAddress,block.timestamp
) (WallStreetJackpot.sol#2601-2608)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (W
allStreetJackpot.sol#2587-2593)
        - swapAndSendDividends(sellTokens) (WallStreetJackpot.sol#2469)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (W
allStreetJackpot.sol#2587-2593)
                - (success) = address(dividendTracker).call{value: dividends}() (WallStreetJackpot.sol#2615)
        External calls sending eth:
        - swapAndSendToFee(marketingTokens) (WallStreetJackpot.sol#2456)
                - address(_marketingWalletAddress).transfer(newBalance) (WallStreetJackpot.sol#2544)
        - swapAndSendToJackpot(jackpotTokens) (WallStreetJackpot.sol#2461)
                - address(address(jackpot)).transfer(newBalance) (WallStreetJackpot.sol#2552)
        - swapAndLiquify(swapTokens) (WallStreetJackpot.sol#2466)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,_marketingWalletAddress,block.timestamp
```

```
        - swapAndLiquify(swapTokens) (WallStreetJackpot.sol#2466)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,_marketingWalletAddress,block.timestamp
) (WallStreetJackpot.sol#2601-2608)
        - swapAndSendDividends(sellTokens) (WallStreetJackpot.sol#2469)
                - (success) = address(dividendTracker).call{value: dividends}() (WallStreetJackpot.sol#2615)
        State variables written after the call(s):
        - swapping = false (WallStreetJackpot.sol#2471)
Reentrancy in WallStreetJackpot._transfer(address,address,uint256) (WallStreetJackpot.sol#2398-2517):
        External calls:
        - swapAndSendToFee(marketingTokens) (WallStreetJackpot.sol#2456)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (W
allStreetJackpot.sol#2587-2593)
        - swapAndSendToJackpot(jackpotTokens) (WallStreetJackpot.sol#2461)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (W
allStreetJackpot.sol#2587-2593)
        - swapAndLiquify(swapTokens) (WallStreetJackpot.sol#2466)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,_marketingWalletAddress,block.timestamp
) (WallStreetJackpot.sol#2601-2608)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (W
allStreetJackpot.sol#2587-2593)
        - swapAndSendDividends(sellTokens) (WallStreetJackpot.sol#2469)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (W
allStreetJackpot.sol#2587-2593)
                - (success) = address(dividendTracker).call{value: dividends}() (WallStreetJackpot.sol#2615)
        - jackpot.jackpot(uniswapV2Pair,from,to,amount) (WallStreetJackpot.sol#2476)
        External calls sending eth:
        - swapAndSendToFee(marketingTokens) (WallStreetJackpot.sol#2456)
                - address(_marketingWalletAddress).transfer(newBalance) (WallStreetJackpot.sol#2544)
        - swapAndSendToJackpot(jackpotTokens) (WallStreetJackpot.sol#2461)
                - address(address(jackpot)).transfer(newBalance) (WallStreetJackpot.sol#2552)
        - swapAndLiquify(swapTokens) (WallStreetJackpot.sol#2466)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,_marketingWalletAddress,block.timestamp
) (WallStreetJackpot.sol#2601-2608)
        - swapAndSendDividends(sellTokens) (WallStreetJackpot.sol#2469)
                - (success) = address(dividendTracker).call{value: dividends}() (WallStreetJackpot.sol#2615)
        State variables written after the call(s):
        - super._transfer(from,address(this),fees) (WallStreetJackpot.sol#2489)
                - _balances[sender] = _balances[sender].sub(amount,ERC20: transfer amount exceeds balance) (WallStreetJackpot.sol#1044-10
```

```
47)
                - _balances[recipient] = _balances[recipient].add(amount) (WallStreetJackpot.sol#1048)
        - super._transfer(from,to,amount) (WallStreetJackpot.sol#2492)
                - _balances[sender] = _balances[sender].sub(amount,ERC20: transfer amount exceeds balance) (WallStreetJackpot.sol#1044-10
47)
                - _balances[recipient] = _balances[recipient].add(amount) (WallStreetJackpot.sol#1048)
Reentrancy in DividendPayingToken._withdrawDividendOfUser(address) (WallStreetJackpot.sol#1824-1850):
        External calls:
        - (success) = user.call{gas: 3000,value: _withdrawableDividend}() (WallStreetJackpot.sol#1834-1837)
        State variables written after the call(s):
        - withdrawnDividends[user] = withdrawnDividends[user].sub(_withdrawableDividend) (WallStreetJackpot.sol#1840-1842)
Reentrancy in Jackpot.claimJackpotReward(uint256) (WallStreetJackpot.sol#1555-1575):
        External calls:
        - (success) = msg.sender.call{value: amount}() (WallStreetJackpot.sol#1565)
        State variables written after the call(s):
        - _isClaimed[round][msg.sender] = true (WallStreetJackpot.sol#1573)
        - _toClaimReward[round] = _toClaimReward[round].sub(amount) (WallStreetJackpot.sol#1571)
        - _totalToClaimReward = _totalToClaimReward.sub(amount) (WallStreetJackpot.sol#1570)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities
```

```
INFO:Detectors:
Jackpot.jackpot(address,address,address,uint256) (WallStreetJackpot.sol#1342-1388) performs a multiplication on the result of a division:
        -tickets = amount.div(_presalePrice).mul(2) (WallStreetJackpot.sol#1370)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
Reentrancy in WallStreetJackpot.updateDividendTracker(address) (WallStreetJackpot.sol#2162-2185):
        External calls:
        - newDividendTracker.excludeFromDividends(address(newDividendTracker)) (WallStreetJackpot.sol#2177)
        - newDividendTracker.excludeFromDividends(address(this)) (WallStreetJackpot.sol#2178)
        - newDividendTracker.excludeFromDividends(owner()) (WallStreetJackpot.sol#2179)
        - newDividendTracker.excludeFromDividends(address(uniswapV2Router)) (WallStreetJackpot.sol#2180)
        State variables written after the call(s):
        - dividendTracker = newDividendTracker (WallStreetJackpot.sol#2184)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
WallStreetJackpot._transfer(address,address,uint256).lastProcessedIndex (WallStreetJackpot.sol#2505) is a local variable never initialize
d
WallStreetJackpot._transfer(address,address,uint256).iterations (WallStreetJackpot.sol#2503) is a local variable never initialized
WallStreetJackpot._transfer(address,address,uint256).claims (WallStreetJackpot.sol#2504) is a local variable never initialized
FixedPoint.mul(FixedPoint.uq112x112,uint256).z (WallStreetJackpot.sol#1642) is a local variable never initialized
Jackpot.jackpot(uint256,uint256,uint256).total (WallStreetJackpot.sol#1428) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
WallStreetJackpot.claim() (WallStreetJackpot.sol#2386-2388) ignores return value by dividendTracker.processAccount(msg.sender,false) (Wal
lStreetJackpot.sol#2387)
WallStreetJackpot._transfer(address,address,uint256) (WallStreetJackpot.sol#2398-2517) ignores return value by dividendTracker.process(ga
s) (WallStreetJackpot.sol#2502-2515)
WallStreetJackpot.addLiquidity(uint256,uint256) (WallStreetJackpot.sol#2596-2609) ignores return value by uniswapV2Router.addLiquidityETH
{value: ethAmount}(address(this),tokenAmount,0,0,_marketingWalletAddress,block.timestamp) (WallStreetJackpot.sol#2601-2608)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
DividendPayingToken.constructor(string,string)._name (WallStreetJackpot.sol#1778) shadows:
        - ERC20._name (WallStreetJackpot.sol#818) (state variable)
DividendPayingToken.constructor(string,string)._symbol (WallStreetJackpot.sol#1778) shadows:
        - ERC20._symbol (WallStreetJackpot.sol#819) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
```

```
INFO:Detectors:
Ownable.constructor().msgSender (WallStreetJackpot.sol#1155) lacks a zero-check on :
                - _owner = msgSender (WallStreetJackpot.sol#1156)
WallStreetJackpot.constructor(address)._uniswapV2Pair (WallStreetJackpot.sol#2055-2056) lacks a zero-check on :
                - uniswapV2Pair = _uniswapV2Pair (WallStreetJackpot.sol#2059)
WallStreetJackpot.updateUniswapV2Router(address)._uniswapV2Pair (WallStreetJackpot.sol#2194-2195) lacks a zero-check on :
                - uniswapV2Pair = _uniswapV2Pair (WallStreetJackpot.sol#2196)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Variable 'WallStreetJackpot._transfer(address,address,uint256).claims (WallStreetJackpot.sol#2504)' in WallStreetJackpot._transfer(addres
s,address,uint256) (WallStreetJackpot.sol#2398-2517) potentially used before declaration: ProcessedDividendTracker(iterations,claims,last
ProcessedIndex,true,gas,tx.origin) (WallStreetJackpot.sol#2507-2514)
Variable 'WallStreetJackpot._transfer(address,address,uint256).iterations (WallStreetJackpot.sol#2503)' in WallStreetJackpot._transfer(ad
dress,address,uint256) (WallStreetJackpot.sol#2398-2517) potentially used before declaration: ProcessedDividendTracker(iterations,claims,
lastProcessedIndex,true,gas,tx.origin) (WallStreetJackpot.sol#2507-2514)
Variable 'WallStreetJackpot._transfer(address,address,uint256).lastProcessedIndex (WallStreetJackpot.sol#2505)' in WallStreetJackpot._tra
nsfer(address,address,uint256) (WallStreetJackpot.sol#2398-2517) potentially used before declaration: ProcessedDividendTracker(iterations
,claims,lastProcessedIndex,true,gas,tx.origin) (WallStreetJackpot.sol#2507-2514)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables
INFO:Detectors:
Reentrancy in WallStreetJackpot._transfer(address,address,uint256) (WallStreetJackpot.sol#2398-2517):
        External calls:
        - swapAndSendToFee(marketingTokens) (WallStreetJackpot.sol#2456)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (W
allStreetJackpot.sol#2587-2593)
        - swapAndSendToJackpot(jackpotTokens) (WallStreetJackpot.sol#2461)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (W
allStreetJackpot.sol#2587-2593)
        External calls sending eth:
        - swapAndSendToFee(marketingTokens) (WallStreetJackpot.sol#2456)
                - address(_marketingWalletAddress).transfer(newBalance) (WallStreetJackpot.sol#2544)
        - swapAndSendToJackpot(jackpotTokens) (WallStreetJackpot.sol#2461)
                - address(address(jackpot)).transfer(newBalance) (WallStreetJackpot.sol#2552)
        State variables written after the call(s):
        - swapAndSendToJackpot(jackpotTokens) (WallStreetJackpot.sol#2461)
                - _allowances[owner][spender] = amount (WallStreetJackpot.sol#1116)
Reentrancy in WallStreetJackpot._transfer(address,address,uint256) (WallStreetJackpot.sol#2398-2517):
        External calls:
```

```
        - swapAndSendToFee(marketingTokens) (WallStreetJackpot.sol#2456)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (W
allStreetJackpot.sol#2587-2593)
        - swapAndSendToJackpot(jackpotTokens) (WallStreetJackpot.sol#2461)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (W
allStreetJackpot.sol#2587-2593)
        - swapAndLiquify(swapTokens) (WallStreetJackpot.sol#2466)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,_marketingWalletAddress,block.timestamp
) (WallStreetJackpot.sol#2601-2608)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (W
allStreetJackpot.sol#2587-2593)
        External calls sending eth:
        - swapAndSendToFee(marketingTokens) (WallStreetJackpot.sol#2456)
                - address(_marketingWalletAddress).transfer(newBalance) (WallStreetJackpot.sol#2544)
        - swapAndSendToJackpot(jackpotTokens) (WallStreetJackpot.sol#2461)
                - address(address(jackpot)).transfer(newBalance) (WallStreetJackpot.sol#2552)
        - swapAndLiquify(swapTokens) (WallStreetJackpot.sol#2466)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,_marketingWalletAddress,block.timestamp
) (WallStreetJackpot.sol#2601-2608)
        State variables written after the call(s):
        - swapAndLiquify(swapTokens) (WallStreetJackpot.sol#2466)
                - _allowances[owner][spender] = amount (WallStreetJackpot.sol#1116)
Reentrancy in WallStreetJackpot._transfer(address,address,uint256) (WallStreetJackpot.sol#2398-2517):
        External calls:
        - swapAndSendToFee(marketingTokens) (WallStreetJackpot.sol#2456)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (W
allStreetJackpot.sol#2587-2593)
        - swapAndSendToJackpot(jackpotTokens) (WallStreetJackpot.sol#2461)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (W
allStreetJackpot.sol#2587-2593)
        - swapAndLiquify(swapTokens) (WallStreetJackpot.sol#2466)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,_marketingWalletAddress,block.timestamp
) (WallStreetJackpot.sol#2601-2608)
```

```
) (WallStreetJackpot.sol#2601-2608)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (W
allStreetJackpot.sol#2587-2593)
        - swapAndSendDividends(sellTokens) (WallStreetJackpot.sol#2469)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (W
allStreetJackpot.sol#2587-2593)
                - (success) = address(dividendTracker).call{value: dividends}() (WallStreetJackpot.sol#2615)
        External calls sending eth:
        - swapAndSendToFee(marketingTokens) (WallStreetJackpot.sol#2456)
                - address(_marketingWalletAddress).transfer(newBalance) (WallStreetJackpot.sol#2544)
        - swapAndSendToJackpot(jackpotTokens) (WallStreetJackpot.sol#2461)
                - address(address(jackpot)).transfer(newBalance) (WallStreetJackpot.sol#2552)
        - swapAndLiquify(swapTokens) (WallStreetJackpot.sol#2466)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,_marketingWalletAddress,block.timestamp
) (WallStreetJackpot.sol#2601-2608)
        - swapAndSendDividends(sellTokens) (WallStreetJackpot.sol#2469)
                - (success) = address(dividendTracker).call{value: dividends}() (WallStreetJackpot.sol#2615)
        State variables written after the call(s):
        - swapAndSendDividends(sellTokens) (WallStreetJackpot.sol#2469)
                - _allowances[owner][spender] = amount (WallStreetJackpot.sol#1116)
Reentrancy in Jackpot.claimJackpotReward(uint256) (WallStreetJackpot.sol#1555-1575):
        External calls:
        - (success) = msg.sender.call{value: amount}() (WallStreetJackpot.sol#1565)
        State variables written after the call(s):
        - _claimedReward[round] = _claimedReward[round].add(amount) (WallStreetJackpot.sol#1568)
        - _totalClaimedReward = _totalClaimedReward.add(amount) (WallStreetJackpot.sol#1567)
Reentrancy in WallStreetJackpot.constructor(address) (WallStreetJackpot.sol#2049-2080):
        External calls:
        - _uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory()).createPair(address(this),_uniswapV2Router.WETH()) (WallStreetJac
kpot.sol#2055-2056)
        State variables written after the call(s):
        - uniswapV2Pair = _uniswapV2Pair (WallStreetJackpot.sol#2059)
        - uniswapV2Router = _uniswapV2Router (WallStreetJackpot.sol#2058)
Reentrancy in WallStreetJackpot.constructor(address) (WallStreetJackpot.sol#2049-2080):
        External calls:
        - _uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory()).createPair(address(this),_uniswapV2Router.WETH()) (WallStreetJac
kpot.sol#2055-2056)
        - _setAutomatedMarketMakerPair(_uniswapV2Pair,true) (WallStreetJackpot.sol#2061)
```

```
                - dividendTracker.excludeFromDividends(pair) (WallStreetJackpot.sol#2281)
        - dividendTracker.excludeFromDividends(address(dividendTracker)) (WallStreetJackpot.sol#2064)
        - dividendTracker.excludeFromDividends(address(this)) (WallStreetJackpot.sol#2065)
        - dividendTracker.excludeFromDividends(owner()) (WallStreetJackpot.sol#2066)
        - dividendTracker.excludeFromDividends(deadWallet) (WallStreetJackpot.sol#2067)
        - dividendTracker.excludeFromDividends(address(_uniswapV2Router)) (WallStreetJackpot.sol#2068)
        State variables written after the call(s):
        - _mint(_marketingWalletAddress,100000000000 * (10 ** 18)) (WallStreetJackpot.sol#2079)
                - _balances[account] = _balances[account].add(amount) (WallStreetJackpot.sol#1067)
        - excludeFromFees(owner(),true) (WallStreetJackpot.sol#2071)
                - _isExcludedFromFees[account] = excluded (WallStreetJackpot.sol#2204)
        - excludeFromFees(_marketingWalletAddress,true) (WallStreetJackpot.sol#2072)
                - _isExcludedFromFees[account] = excluded (WallStreetJackpot.sol#2204)
        - excludeFromFees(address(this),true) (WallStreetJackpot.sol#2073)
                - _isExcludedFromFees[account] = excluded (WallStreetJackpot.sol#2204)
        - _mint(_marketingWalletAddress,100000000000 * (10 ** 18)) (WallStreetJackpot.sol#2079)
                - _totalSupply = _totalSupply.add(amount) (WallStreetJackpot.sol#1066)
Reentrancy in WallStreetJackpotDividendTracker.processAccount(address,bool) (WallStreetJackpot.sol#2869-2883):
        External calls:
        - amount = _withdrawDividendOfUser(account) (WallStreetJackpot.sol#2874)
                - (success) = user.call{gas: 3000,value: _withdrawableDividend}() (WallStreetJackpot.sol#1834-1837)
        State variables written after the call(s):
        - lastClaimTimes[account] = block.timestamp (WallStreetJackpot.sol#2877)
Reentrancy in WallStreetJackpot.setDxSaleAddress(address,address) (WallStreetJackpot.sol#2519-2530):
        External calls:
        - dividendTracker.excludeFromDividends(dxRouter) (WallStreetJackpot.sol#2523)
        - dividendTracker.excludeFromDividends(presaleRouter) (WallStreetJackpot.sol#2524)
        - jackpot.setDxsale(dxRouter,presaleRouter) (WallStreetJackpot.sol#2526)
        State variables written after the call(s):
        - excludeFromFees(dxRouter,true) (WallStreetJackpot.sol#2528)
                - _isExcludedFromFees[account] = excluded (WallStreetJackpot.sol#2204)
        - excludeFromFees(presaleRouter,true) (WallStreetJackpot.sol#2529)
                - _isExcludedFromFees[account] = excluded (WallStreetJackpot.sol#2204)
Reentrancy in WallStreetJackpot.swapAndLiquify(uint256) (WallStreetJackpot.sol#2555-2576):
        External calls:
        - swapTokensForEth(half) (WallStreetJackpot.sol#2567)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (W
allStreetJackpot.sol#2587-2593)
```

```
allStreetJackpot.sol#2587-2593)
        - addLiquidity(otherHalf,newBalance) (WallStreetJackpot.sol#2573)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,_marketingWalletAddress,block.timestamp
) (WallStreetJackpot.sol#2601-2608)
        External calls sending eth:
        - addLiquidity(otherHalf,newBalance) (WallStreetJackpot.sol#2573)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,_marketingWalletAddress,block.timestamp
) (WallStreetJackpot.sol#2601-2608)
        State variables written after the call(s):
        - addLiquidity(otherHalf,newBalance) (WallStreetJackpot.sol#2573)
                - _allowances[owner][spender] = amount (WallStreetJackpot.sol#1116)
Reentrancy in WallStreetJackpot.updateUniswapV2Router(address) (WallStreetJackpot.sol#2187-2197):
        External calls:
        - _uniswapV2Pair = IUniswapV2Factory(uniswapV2Router.factory()).createPair(address(this),uniswapV2Router.WETH()) (WallStreetJackp
ot.sol#2194-2195)
        State variables written after the call(s):
        - uniswapV2Pair = _uniswapV2Pair (WallStreetJackpot.sol#2196)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in WallStreetJackpot._setAutomatedMarketMakerPair(address,bool) (WallStreetJackpot.sol#2273-2285):
        External calls:
        - dividendTracker.excludeFromDividends(pair) (WallStreetJackpot.sol#2281)
        Event emitted after the call(s):
        - SetAutomatedMarketMakerPair(pair,value) (WallStreetJackpot.sol#2284)
Reentrancy in WallStreetJackpot._transfer(address,address,uint256) (WallStreetJackpot.sol#2398-2517):
        External calls:
        - swapAndSendToFee(marketingTokens) (WallStreetJackpot.sol#2456)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (W
allStreetJackpot.sol#2587-2593)
        - swapAndSendToJackpot(jackpotTokens) (WallStreetJackpot.sol#2461)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (W
allStreetJackpot.sol#2587-2593)
```

```
        External calls sending eth:
        - swapAndSendToFee(marketingTokens) (WallStreetJackpot.sol#2456)
                - address(_marketingWalletAddress).transfer(newBalance) (WallStreetJackpot.sol#2544)
        - swapAndSendToJackpot(jackpotTokens) (WallStreetJackpot.sol#2461)
                - address(address(jackpot)).transfer(newBalance) (WallStreetJackpot.sol#2552)
        Event emitted after the call(s):
        - Approval(owner,spender,amount) (WallStreetJackpot.sol#1117)
                - swapAndSendToJackpot(jackpotTokens) (WallStreetJackpot.sol#2461)
Reentrancy in WallStreetJackpot._transfer(address,address,uint256) (WallStreetJackpot.sol#2398-2517):
        External calls:
        - swapAndSendToFee(marketingTokens) (WallStreetJackpot.sol#2456)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (W
allStreetJackpot.sol#2587-2593)
        - swapAndSendToJackpot(jackpotTokens) (WallStreetJackpot.sol#2461)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (W
allStreetJackpot.sol#2587-2593)
        - swapAndLiquify(swapTokens) (WallStreetJackpot.sol#2466)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,_marketingWalletAddress,block.timestamp
) (WallStreetJackpot.sol#2601-2608)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (W
allStreetJackpot.sol#2587-2593)
        External calls sending eth:
        - swapAndSendToFee(marketingTokens) (WallStreetJackpot.sol#2456)
                - address(_marketingWalletAddress).transfer(newBalance) (WallStreetJackpot.sol#2544)
        - swapAndSendToJackpot(jackpotTokens) (WallStreetJackpot.sol#2461)
                - address(address(jackpot)).transfer(newBalance) (WallStreetJackpot.sol#2552)
        - swapAndLiquify(swapTokens) (WallStreetJackpot.sol#2466)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,_marketingWalletAddress,block.timestamp
) (WallStreetJackpot.sol#2601-2608)
        Event emitted after the call(s):
        - Approval(owner,spender,amount) (WallStreetJackpot.sol#1117)
                - swapAndLiquify(swapTokens) (WallStreetJackpot.sol#2466)
        - SwapAndLiquify(half,newBalance,otherHalf) (WallStreetJackpot.sol#2575)
                - swapAndLiquify(swapTokens) (WallStreetJackpot.sol#2466)
Reentrancy in WallStreetJackpot._transfer(address,address,uint256) (WallStreetJackpot.sol#2398-2517):
        External calls:
        - swapAndSendToFee(marketingTokens) (WallStreetJackpot.sol#2456)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (W
```

```
allStreetJackpot.sol#2587-2593)
        - swapAndSendToJackpot(jackpotTokens) (WallStreetJackpot.sol#2461)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (W
allStreetJackpot.sol#2587-2593)
        - swapAndLiquify(swapTokens) (WallStreetJackpot.sol#2466)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,_marketingWalletAddress,block.timestamp
) (WallStreetJackpot.sol#2601-2608)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (W
allStreetJackpot.sol#2587-2593)
        - swapAndSendDividends(sellTokens) (WallStreetJackpot.sol#2469)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (W
allStreetJackpot.sol#2587-2593)
                - (success) = address(dividendTracker).call{value: dividends}() (WallStreetJackpot.sol#2615)
        External calls sending eth:
        - swapAndSendToFee(marketingTokens) (WallStreetJackpot.sol#2456)
                - address(_marketingWalletAddress).transfer(newBalance) (WallStreetJackpot.sol#2544)
        - swapAndSendToJackpot(jackpotTokens) (WallStreetJackpot.sol#2461)
                - address(address(jackpot)).transfer(newBalance) (WallStreetJackpot.sol#2552)
        - swapAndLiquify(swapTokens) (WallStreetJackpot.sol#2466)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,_marketingWalletAddress,block.timestamp
) (WallStreetJackpot.sol#2601-2608)
        - swapAndSendDividends(sellTokens) (WallStreetJackpot.sol#2469)
                - (success) = address(dividendTracker).call{value: dividends}() (WallStreetJackpot.sol#2615)
        Event emitted after the call(s):
        - Approval(owner,spender,amount) (WallStreetJackpot.sol#1117)
                - swapAndSendDividends(sellTokens) (WallStreetJackpot.sol#2469)
        - SendDividends(tokens,dividends) (WallStreetJackpot.sol#2618)
                - swapAndSendDividends(sellTokens) (WallStreetJackpot.sol#2469)
Reentrancy in WallStreetJackpot._transfer(address,address,uint256) (WallStreetJackpot.sol#2398-2517):
        External calls:
        - swapAndSendToFee(marketingTokens) (WallStreetJackpot.sol#2456)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (W
allStreetJackpot.sol#2587-2593)
        - swapAndSendToJackpot(jackpotTokens) (WallStreetJackpot.sol#2461)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (W
allStreetJackpot.sol#2587-2593)
        - swapAndLiquify(swapTokens) (WallStreetJackpot.sol#2466)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,_marketingWalletAddress,block.timestamp
```

```
) (WallStreetJackpot.sol#2601-2608)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (W
allStreetJackpot.sol#2587-2593)
        - swapAndSendDividends(sellTokens) (WallStreetJackpot.sol#2469)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (W
allStreetJackpot.sol#2587-2593)
                - (success) = address(dividendTracker).call{value: dividends}() (WallStreetJackpot.sol#2615)
        - jackpot.jackpot(uniswapV2Pair,from,to,amount) (WallStreetJackpot.sol#2476)
        External calls sending eth:
        - swapAndSendToFee(marketingTokens) (WallStreetJackpot.sol#2456)
                - address(_marketingWalletAddress).transfer(newBalance) (WallStreetJackpot.sol#2544)
        - swapAndSendToJackpot(jackpotTokens) (WallStreetJackpot.sol#2461)
                - address(address(jackpot)).transfer(newBalance) (WallStreetJackpot.sol#2552)
        - swapAndLiquify(swapTokens) (WallStreetJackpot.sol#2466)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,_marketingWalletAddress,block.timestamp
) (WallStreetJackpot.sol#2601-2608)
        - swapAndSendDividends(sellTokens) (WallStreetJackpot.sol#2469)
                - (success) = address(dividendTracker).call{value: dividends}() (WallStreetJackpot.sol#2615)
        Event emitted after the call(s):
        - Transfer(sender,recipient,amount) (WallStreetJackpot.sol#1049)
                - super._transfer(from,address(this),fees) (WallStreetJackpot.sol#2489)
        - Transfer(sender,recipient,amount) (WallStreetJackpot.sol#1049)
                - super._transfer(from,to,amount) (WallStreetJackpot.sol#2492)
Reentrancy in WallStreetJackpot._transfer(address,address,uint256) (WallStreetJackpot.sol#2398-2517):
        External calls:
        - swapAndSendToFee(marketingTokens) (WallStreetJackpot.sol#2456)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (W
allStreetJackpot.sol#2587-2593)
        - swapAndSendToJackpot(jackpotTokens) (WallStreetJackpot.sol#2461)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (W
allStreetJackpot.sol#2587-2593)
        - swapAndLiquify(swapTokens) (WallStreetJackpot.sol#2466)
```

```
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,_marketingWalletAddress,block.timestamp
) (WallStreetJackpot.sol#2601-2608)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (W
allStreetJackpot.sol#2587-2593)
        - swapAndSendDividends(sellTokens) (WallStreetJackpot.sol#2469)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (W
allStreetJackpot.sol#2587-2593)
                - (success) = address(dividendTracker).call{value: dividends}() (WallStreetJackpot.sol#2615)
        - jackpot.jackpot(uniswapV2Pair,from,to,amount) (WallStreetJackpot.sol#2476)
        - dividendTracker.setBalance(address(from),balanceOf(from)) (WallStreetJackpot.sol#2494-2496)
        - dividendTracker.setBalance(address(to),balanceOf(to)) (WallStreetJackpot.sol#2497)
        - dividendTracker.process(gas) (WallStreetJackpot.sol#2502-2515)
        External calls sending eth:
        - swapAndSendToFee(marketingTokens) (WallStreetJackpot.sol#2456)
                - address(_marketingWalletAddress).transfer(newBalance) (WallStreetJackpot.sol#2544)
        - swapAndSendToJackpot(jackpotTokens) (WallStreetJackpot.sol#2461)
                - address(address(jackpot)).transfer(newBalance) (WallStreetJackpot.sol#2552)
        - swapAndLiquify(swapTokens) (WallStreetJackpot.sol#2466)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,_marketingWalletAddress,block.timestamp
) (WallStreetJackpot.sol#2601-2608)
        - swapAndSendDividends(sellTokens) (WallStreetJackpot.sol#2469)
                - (success) = address(dividendTracker).call{value: dividends}() (WallStreetJackpot.sol#2615)
        Event emitted after the call(s):
        - ProcessedDividendTracker(iterations,claims,lastProcessedIndex,true,gas,tx.origin) (WallStreetJackpot.sol#2507-2514)
Reentrancy in WallStreetJackpot.constructor(address) (WallStreetJackpot.sol#2049-2080):
        External calls:
        - _uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory()).createPair(address(this),_uniswapV2Router.WETH()) (WallStreetJac
kpot.sol#2055-2056)
        - _setAutomatedMarketMakerPair(_uniswapV2Pair,true) (WallStreetJackpot.sol#2061)
                - dividendTracker.excludeFromDividends(pair) (WallStreetJackpot.sol#2281)
        Event emitted after the call(s):
        - SetAutomatedMarketMakerPair(pair,value) (WallStreetJackpot.sol#2284)
                - _setAutomatedMarketMakerPair(_uniswapV2Pair,true) (WallStreetJackpot.sol#2061)
Reentrancy in WallStreetJackpot.constructor(address) (WallStreetJackpot.sol#2049-2080):
        External calls:
        - _uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory()).createPair(address(this),_uniswapV2Router.WETH()) (WallStreetJac
kpot.sol#2055-2056)
        - _setAutomatedMarketMakerPair(_uniswapV2Pair,true) (WallStreetJackpot.sol#2061)
```

```
                - dividendTracker.excludeFromDividends(pair) (WallStreetJackpot.sol#2281)
        - dividendTracker.excludeFromDividends(address(dividendTracker)) (WallStreetJackpot.sol#2064)
        - dividendTracker.excludeFromDividends(address(this)) (WallStreetJackpot.sol#2065)
        - dividendTracker.excludeFromDividends(owner()) (WallStreetJackpot.sol#2066)
        - dividendTracker.excludeFromDividends(deadWallet) (WallStreetJackpot.sol#2067)
        - dividendTracker.excludeFromDividends(address(_uniswapV2Router)) (WallStreetJackpot.sol#2068)
        Event emitted after the call(s):
        - ExcludeFromFees(account,excluded) (WallStreetJackpot.sol#2206)
                - excludeFromFees(_marketingWalletAddress,true) (WallStreetJackpot.sol#2072)
        - ExcludeFromFees(account,excluded) (WallStreetJackpot.sol#2206)
                - excludeFromFees(address(this),true) (WallStreetJackpot.sol#2073)
        - ExcludeFromFees(account,excluded) (WallStreetJackpot.sol#2206)
                - excludeFromFees(owner(),true) (WallStreetJackpot.sol#2071)
        - Transfer(address(0),account,amount) (WallStreetJackpot.sol#1068)
                - _mint(_marketingWalletAddress,100000000000 * (10 ** 18)) (WallStreetJackpot.sol#2079)
Reentrancy in WallStreetJackpotDividendTracker.processAccount(address,bool) (WallStreetJackpot.sol#2869-2883):
        External calls:
        - amount = _withdrawDividendOfUser(account) (WallStreetJackpot.sol#2874)
                - (success) = user.call{gas: 3000,value: _withdrawableDividend}() (WallStreetJackpot.sol#1834-1837)
        Event emitted after the call(s):
        - Claim(account,amount,automatic) (WallStreetJackpot.sol#2878)
Reentrancy in WallStreetJackpot.processDividendTracker(uint256) (WallStreetJackpot.sol#2370-2384):
        External calls:
        - (iterations,claims,lastProcessedIndex) = dividendTracker.process(gas) (WallStreetJackpot.sol#2371-2375)
        Event emitted after the call(s):
        - ProcessedDividendTracker(iterations,claims,lastProcessedIndex,false,gas,tx.origin) (WallStreetJackpot.sol#2376-2383)
Reentrancy in WallStreetJackpot.setDxSaleAddress(address,address) (WallStreetJackpot.sol#2519-2530):
        External calls:
        - dividendTracker.excludeFromDividends(dxRouter) (WallStreetJackpot.sol#2523)
        - dividendTracker.excludeFromDividends(presaleRouter) (WallStreetJackpot.sol#2524)
        - jackpot.setDxsale(dxRouter,presaleRouter) (WallStreetJackpot.sol#2526)
        Event emitted after the call(s):
        - ExcludeFromFees(account,excluded) (WallStreetJackpot.sol#2206)
                - excludeFromFees(presaleRouter,true) (WallStreetJackpot.sol#2529)
        - ExcludeFromFees(account,excluded) (WallStreetJackpot.sol#2206)
                - excludeFromFees(dxRouter,true) (WallStreetJackpot.sol#2528)
Reentrancy in WallStreetJackpot.swapAndLiquify(uint256) (WallStreetJackpot.sol#2555-2576):
        External calls:
```

```
        - swapTokensForEth(half) (WallStreetJackpot.sol#2567)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (W
allStreetJackpot.sol#2587-2593)
        - addLiquidity(otherHalf,newBalance) (WallStreetJackpot.sol#2573)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,_marketingWalletAddress,block.timestamp
) (WallStreetJackpot.sol#2601-2608)
        External calls sending eth:
        - addLiquidity(otherHalf,newBalance) (WallStreetJackpot.sol#2573)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,_marketingWalletAddress,block.timestamp
) (WallStreetJackpot.sol#2601-2608)
        Event emitted after the call(s):
        - Approval(owner,spender,amount) (WallStreetJackpot.sol#1117)
                - addLiquidity(otherHalf,newBalance) (WallStreetJackpot.sol#2573)
        - SwapAndLiquify(half,newBalance,otherHalf) (WallStreetJackpot.sol#2575)
Reentrancy in WallStreetJackpot.swapAndSendDividends(uint256) (WallStreetJackpot.sol#2611-2620):
        External calls:
        - swapTokensForEth(tokens) (WallStreetJackpot.sol#2613)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (W
allStreetJackpot.sol#2587-2593)
        - (success) = address(dividendTracker).call{value: dividends}() (WallStreetJackpot.sol#2615)
        External calls sending eth:
        - (success) = address(dividendTracker).call{value: dividends}() (WallStreetJackpot.sol#2615)
        Event emitted after the call(s):
        - SendDividends(tokens,dividends) (WallStreetJackpot.sol#2618)
Reentrancy in WallStreetJackpot.updateDividendTracker(address) (WallStreetJackpot.sol#2162-2185):
        External calls:
        - newDividendTracker.excludeFromDividends(address(newDividendTracker)) (WallStreetJackpot.sol#2177)
        - newDividendTracker.excludeFromDividends(address(this)) (WallStreetJackpot.sol#2178)
        - newDividendTracker.excludeFromDividends(owner()) (WallStreetJackpot.sol#2179)
        - newDividendTracker.excludeFromDividends(address(uniswapV2Router)) (WallStreetJackpot.sol#2180)
        Event emitted after the call(s):
        - UpdateDividendTracker(newAddress,address(dividendTracker)) (WallStreetJackpot.sol#2182)
```

```
Reentrancy in Jackpot.withdrawEth() (WallStreetJackpot.sol#1577-1584):
        External calls:
        - (success) = address(_marketWallet).call{value: amount}() (WallStreetJackpot.sol#1580)
        Event emitted after the call(s):
        - WithdrawETH(_marketWallet,amount) (WallStreetJackpot.sol#1582)
Reentrancy in Jackpot.withdrawEth(uint256) (WallStreetJackpot.sol#1586-1592):
        External calls:
        - (success) = address(_marketWallet).call{value: amount}() (WallStreetJackpot.sol#1588)
        Event emitted after the call(s):
        - WithdrawETH(_marketWallet,amount) (WallStreetJackpot.sol#1590)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Jackpot.jackpot(address,address,address,uint256) (WallStreetJackpot.sol#1342-1388) uses timestamp for comparisons
        Dangerous comparisons:
        - totalReward >= _minPrize && _lastJackpotTS > 0 && (block.timestamp - _lastJackpotTS) >= _jackpotPeriod && ! _jackpoting (WallSt
reetJackpot.sol#1354-1357)
Jackpot.jackpot(uint256,uint256,uint256) (WallStreetJackpot.sol#1421-1446) uses timestamp for comparisons
        Dangerous comparisons:
        - total >= target (WallStreetJackpot.sol#1438)
WallStreetJackpotDividendTracker.getAccount(address) (WallStreetJackpot.sol#2707-2754) uses timestamp for comparisons
        Dangerous comparisons:
        - nextClaimTime > block.timestamp (WallStreetJackpot.sol#2751-2753)
WallStreetJackpotDividendTracker.canAutoClaim(uint256) (WallStreetJackpot.sol#2788-2794) uses timestamp for comparisons
        Dangerous comparisons:
        - lastClaimTime > block.timestamp (WallStreetJackpot.sol#2789)
        - block.timestamp.sub(lastClaimTime) >= claimWait (WallStreetJackpot.sol#2793)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Babylonian.sqrt(uint256) (WallStreetJackpot.sol#237-249) is never used and should be removed
Context._msgData() (WallStreetJackpot.sol#779-782) is never used and should be removed
DividendPayingToken._transfer(address,address,uint256) (WallStreetJackpot.sol#1907-1922) is never used and should be removed
FixedPoint.decode(FixedPoint.uq112x112) (WallStreetJackpot.sol#1662-1664) is never used and should be removed
FixedPoint.div(FixedPoint.uq112x112,uint112) (WallStreetJackpot.sol#1626-1633) is never used and should be removed
FixedPoint.encode(uint112) (WallStreetJackpot.sol#1616-1618) is never used and should be removed
FixedPoint.encode144(uint144) (WallStreetJackpot.sol#1621-1623) is never used and should be removed
FixedPoint.reciprocal(FixedPoint.uq112x112) (WallStreetJackpot.sol#1672-1679) is never used and should be removed
FixedPoint.sqrt(FixedPoint.uq112x112) (WallStreetJackpot.sol#1682-1688) is never used and should be removed
SafeMath.mod(uint256,uint256) (WallStreetJackpot.sol#135-137) is never used and should be removed
```

```
SafeMath.mod(uint256,uint256) (WallStreetJackpot.sol#135-137) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (WallStreetJackpot.sol#151-158) is never used and should be removed
SafeMathInt.abs(int256) (WallStreetJackpot.sol#213-216) is never used and should be removed
SafeMathInt.div(int256,int256) (WallStreetJackpot.sol#184-190) is never used and should be removed
SafeMathInt.mul(int256,int256) (WallStreetJackpot.sol#172-179) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
WallStreetJackpot.totalFees (WallStreetJackpot.sol#1984-1985) is set pre-construction with a non-constant function or state variable:
        - BNBRewardsFee.add(liquidityFee).add(marketingFee).add(jackpotFee)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#function-initializing-state-variables
INFO:Detectors:
Low level call in Jackpot.claimJackpotReward(uint256) (WallStreetJackpot.sol#1555-1575):
        - (success) = msg.sender.call{value: amount}() (WallStreetJackpot.sol#1565)
Low level call in Jackpot.withdrawEth() (WallStreetJackpot.sol#1577-1584):
        - (success) = address(_marketWallet).call{value: amount}() (WallStreetJackpot.sol#1580)
Low level call in Jackpot.withdrawEth(uint256) (WallStreetJackpot.sol#1586-1592):
        - (success) = address(_marketWallet).call{value: amount}() (WallStreetJackpot.sol#1588)
Low level call in DividendPayingToken._withdrawDividendOfUser(address) (WallStreetJackpot.sol#1824-1850):
        - (success) = user.call{gas: 3000,value: _withdrawableDividend}() (WallStreetJackpot.sol#1834-1837)
Low level call in WallStreetJackpot.swapAndSendDividends(uint256) (WallStreetJackpot.sol#2611-2620):
        - (success) = address(dividendTracker).call{value: dividends}() (WallStreetJackpot.sol#2615)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Function IUniswapV2Pair.DOMAIN_SEPARATOR() (WallStreetJackpot.sol#388) is not in mixedCase
Function IUniswapV2Pair.PERMIT_TYPEHASH() (WallStreetJackpot.sol#390) is not in mixedCase
Function IUniswapV2Pair.MINIMUM_LIQUIDITY() (WallStreetJackpot.sol#421) is not in mixedCase
Function IUniswapV2Router01.WETH() (WallStreetJackpot.sol#468) is not in mixedCase
Parameter Jackpot.random(uint256)._length (WallStreetJackpot.sol#1501) is not in mixedCase
Variable Jackpot._marketWallet (WallStreetJackpot.sol#1207) is not in mixedCase
Variable Jackpot._deadWallet (WallStreetJackpot.sol#1208) is not in mixedCase
Variable Jackpot._currentRound (WallStreetJackpot.sol#1210) is not in mixedCase
Variable Jackpot._totalTickets (WallStreetJackpot.sol#1211) is not in mixedCase
Variable Jackpot._totalToClaimReward (WallStreetJackpot.sol#1212) is not in mixedCase
Variable Jackpot._totalClaimedReward (WallStreetJackpot.sol#1213) is not in mixedCase
Variable Jackpot._minPrize (WallStreetJackpot.sol#1232) is not in mixedCase
Variable Jackpot._maxPrize (WallStreetJackpot.sol#1233) is not in mixedCase
Variable Jackpot._totalPrize (WallStreetJackpot.sol#1234) is not in mixedCase
Variable Jackpot._firstPrize (WallStreetJackpot.sol#1236) is not in mixedCase
```

```
Variable Jackpot._firstPrize (WallStreetJackpot.sol#1236) is not in mixedCase
Variable Jackpot._secondPrize (WallStreetJackpot.sol#1237) is not in mixedCase
Variable Jackpot._thirdPrize (WallStreetJackpot.sol#1238) is not in mixedCase
Variable Jackpot._firstPrizeOdds (WallStreetJackpot.sol#1240) is not in mixedCase
Variable Jackpot._secondPrizeOdds (WallStreetJackpot.sol#1241) is not in mixedCase
Variable Jackpot._thirdPrizeOdds (WallStreetJackpot.sol#1242) is not in mixedCase
Variable Jackpot._lastJackpotTS (WallStreetJackpot.sol#1244) is not in mixedCase
Variable Jackpot._jackpotPeriod (WallStreetJackpot.sol#1245) is not in mixedCase
Variable Jackpot._jackpotThreshold (WallStreetJackpot.sol#1246) is not in mixedCase
Variable Jackpot._isBasedOnToken (WallStreetJackpot.sol#1250) is not in mixedCase
Variable Jackpot._tokensAmont (WallStreetJackpot.sol#1251) is not in mixedCase
Variable Jackpot._enableSubTickets (WallStreetJackpot.sol#1253) is not in mixedCase
Variable Jackpot._presalePrice (WallStreetJackpot.sol#1255) is not in mixedCase
Struct FixedPoint.uq112x112 (WallStreetJackpot.sol#1601-1603) is not in CapWords
Struct FixedPoint.uq144x112 (WallStreetJackpot.sol#1607-1609) is not in CapWords
Parameter DividendPayingToken.dividendOf(address)._owner (WallStreetJackpot.sol#1855) is not in mixedCase
Parameter DividendPayingToken.withdrawableDividendOf(address)._owner (WallStreetJackpot.sol#1862) is not in mixedCase
Parameter DividendPayingToken.withdrawnDividendOf(address)._owner (WallStreetJackpot.sol#1874) is not in mixedCase
Parameter DividendPayingToken.accumulativeDividendOf(address)._owner (WallStreetJackpot.sol#1888) is not in mixedCase
Constant DividendPayingToken.magnitude (WallStreetJackpot.sol#1758) is not in UPPER_CASE_WITH_UNDERSCORES
Variable WallStreetJackpot._marketingWalletAddress (WallStreetJackpot.sol#1973-1974) is not in mixedCase
Variable WallStreetJackpot._isBlacklisted (WallStreetJackpot.sol#1978) is not in mixedCase
Variable WallStreetJackpot.BNBRewardsFee (WallStreetJackpot.sol#1980) is not in mixedCase
Variable WallStreetJackpot._jackpotEnabled (WallStreetJackpot.sol#1997) is not in mixedCase
Variable WallStreetJackpot._snipeBlockAmount (WallStreetJackpot.sol#2001) is not in mixedCase
Variable WallStreetJackpot._tradingBlock (WallStreetJackpot.sol#2002) is not in mixedCase
Variable WallStreetJackpot._sniperProtection (WallStreetJackpot.sol#2003) is not in mixedCase
Parameter WallStreetJackpotDividendTracker.getAccount(address)._account (WallStreetJackpot.sol#2707) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```

```
INFO:Detectors:
Redundant expression "this (WallStreetJackpot.sol#780)" inContext (WallStreetJackpot.sol#774-783)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
Reentrancy in WallStreetJackpot._transfer(address,address,uint256) (WallStreetJackpot.sol#2398-2517):
        External calls:
        - swapAndSendToFee(marketingTokens) (WallStreetJackpot.sol#2456)
                - address(_marketingWalletAddress).transfer(newBalance) (WallStreetJackpot.sol#2544)
        - swapAndSendToJackpot(jackpotTokens) (WallStreetJackpot.sol#2461)
                - address(address(jackpot)).transfer(newBalance) (WallStreetJackpot.sol#2552)
        State variables written after the call(s):
        - swapAndSendToJackpot(jackpotTokens) (WallStreetJackpot.sol#2461)
                - _allowances[owner][spender] = amount (WallStreetJackpot.sol#1116)
        Event emitted after the call(s):
        - Approval(owner,spender,amount) (WallStreetJackpot.sol#1117)
                - swapAndSendToJackpot(jackpotTokens) (WallStreetJackpot.sol#2461)
Reentrancy in WallStreetJackpot._transfer(address,address,uint256) (WallStreetJackpot.sol#2398-2517):
        External calls:
        - swapAndSendToFee(marketingTokens) (WallStreetJackpot.sol#2456)
                - address(_marketingWalletAddress).transfer(newBalance) (WallStreetJackpot.sol#2544)
        - swapAndSendToJackpot(jackpotTokens) (WallStreetJackpot.sol#2461)
                - address(address(jackpot)).transfer(newBalance) (WallStreetJackpot.sol#2552)
        External calls sending eth:
        - swapAndSendToFee(marketingTokens) (WallStreetJackpot.sol#2456)
                - address(_marketingWalletAddress).transfer(newBalance) (WallStreetJackpot.sol#2544)
        - swapAndSendToJackpot(jackpotTokens) (WallStreetJackpot.sol#2461)
                - address(address(jackpot)).transfer(newBalance) (WallStreetJackpot.sol#2552)
        - swapAndLiquify(swapTokens) (WallStreetJackpot.sol#2466)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,_marketingWalletAddress,block.timestamp
) (WallStreetJackpot.sol#2601-2608)
        State variables written after the call(s):
        - swapAndLiquify(swapTokens) (WallStreetJackpot.sol#2466)
                - _allowances[owner][spender] = amount (WallStreetJackpot.sol#1116)
        Event emitted after the call(s):
        - Approval(owner,spender,amount) (WallStreetJackpot.sol#1117)
                - swapAndLiquify(swapTokens) (WallStreetJackpot.sol#2466)
        - SwapAndLiquify(half,newBalance,otherHalf) (WallStreetJackpot.sol#2575)
                - swapAndLiquify(swapTokens) (WallStreetJackpot.sol#2466)
```

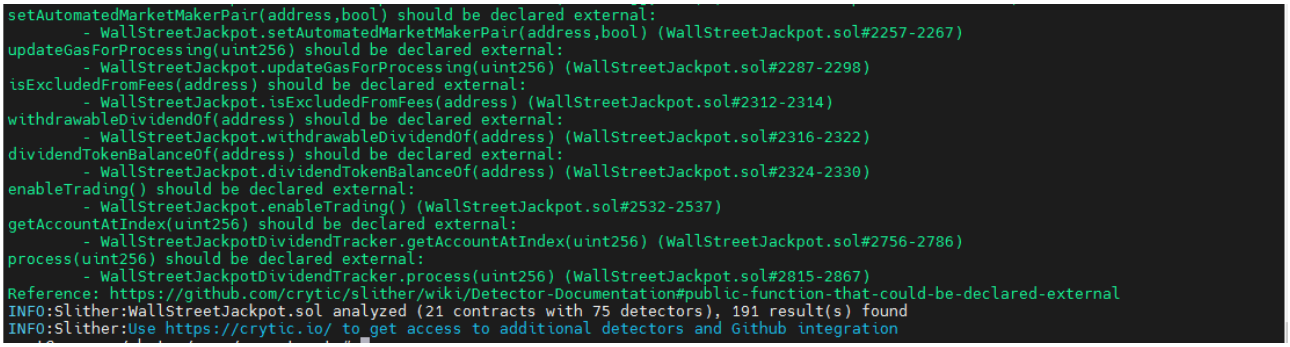
```
Reentrancy in WallStreetJackpot._transfer(address,address,uint256) (WallStreetJackpot.sol#2398-2517):
        External calls:
        - swapAndSendToFee(marketingTokens) (WallStreetJackpot.sol#2456)
                - address(_marketingWalletAddress).transfer(newBalance) (WallStreetJackpot.sol#2544)
        - swapAndSendToJackpot(jackpotTokens) (WallStreetJackpot.sol#2461)
                - address(address(jackpot)).transfer(newBalance) (WallStreetJackpot.sol#2552)
        External calls sending eth:
        - swapAndSendToFee(marketingTokens) (WallStreetJackpot.sol#2456)
                - address(_marketingWalletAddress).transfer(newBalance) (WallStreetJackpot.sol#2544)
        - swapAndSendToJackpot(jackpotTokens) (WallStreetJackpot.sol#2461)
                - address(address(jackpot)).transfer(newBalance) (WallStreetJackpot.sol#2552)
        - swapAndLiquify(swapTokens) (WallStreetJackpot.sol#2466)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,_marketingWalletAddress,block.timestamp
) (WallStreetJackpot.sol#2601-2608)
        - swapAndSendDividends(sellTokens) (WallStreetJackpot.sol#2469)
                - (success) = address(dividendTracker).call{value: dividends}() (WallStreetJackpot.sol#2615)
        State variables written after the call(s):
        - swapAndSendDividends(sellTokens) (WallStreetJackpot.sol#2469)
                - _allowances[owner][spender] = amount (WallStreetJackpot.sol#1116)
        - super._transfer(from,address(this),fees) (WallStreetJackpot.sol#2489)
                - _balances[sender] = _balances[sender].sub(amount,ERC20: transfer amount exceeds balance) (WallStreetJackpot.sol#1044-10
47)
                - _balances[recipient] = _balances[recipient].add(amount) (WallStreetJackpot.sol#1048)
        - super._transfer(from,to,amount) (WallStreetJackpot.sol#2492)
                - _balances[sender] = _balances[sender].sub(amount,ERC20: transfer amount exceeds balance) (WallStreetJackpot.sol#1044-10
47)
                - _balances[recipient] = _balances[recipient].add(amount) (WallStreetJackpot.sol#1048)
        - swapping = false (WallStreetJackpot.sol#2471)
        Event emitted after the call(s):
        - Approval(owner,spender,amount) (WallStreetJackpot.sol#1117)
                - swapAndSendDividends(sellTokens) (WallStreetJackpot.sol#2469)
        - ProcessedDividendTracker(iterations,claims,lastProcessedIndex,true,gas,tx.origin) (WallStreetJackpot.sol#2507-2514)
        - SendDividends(tokens,dividends) (WallStreetJackpot.sol#2618)
                - swapAndSendDividends(sellTokens) (WallStreetJackpot.sol#2469)
        - Transfer(sender,recipient,amount) (WallStreetJackpot.sol#1049)
                - super._transfer(from,address(this),fees) (WallStreetJackpot.sol#2489)
        - Transfer(sender,recipient,amount) (WallStreetJackpot.sol#1049)
                - super._transfer(from,to,amount) (WallStreetJackpot.sol#2492)
```

```
INFO:Detectors:
Variable IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountADesired (WallStreetJackp
ot.sol#473) is too similar to IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountBDes
ired (WallStreetJackpot.sol#474)
Variable WallStreetJackpot.swapTokensForEth(uint256).tokenAmount (WallStreetJackpot.sol#2578) is too similar to WallStreetJackpot.setToke
nsAmont(uint256).tokensAmont (WallStreetJackpot.sol#2117)
Variable WallStreetJackpot.addLiquidity(uint256,uint256).tokenAmount (WallStreetJackpot.sol#2596) is too similar to WallStreetJackpot.set
TokensAmont(uint256).tokensAmont (WallStreetJackpot.sol#2117)
Variable DividendPayingToken._withdrawDividendOfUser(address)._withdrawableDividend (WallStreetJackpot.sol#1828) is too similar to WallSt
reetJackpotDividendTracker.getAccount(address).withdrawableDividends (WallStreetJackpot.sol#2714)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar
INFO:Detectors:
Jackpot.slitherConstructorVariables() (WallStreetJackpot.sol#1202-1595) uses literals with too many digits:
        - _deadWallet = 0x000000000000000000000000000000000000dEaD (WallStreetJackpot.sol#1208)
Jackpot.slitherConstructorVariables() (WallStreetJackpot.sol#1202-1595) uses literals with too many digits:
        - _tokensAmont = 1000000 * (10 ** 18) (WallStreetJackpot.sol#1251)
Jackpot.slitherConstructorVariables() (WallStreetJackpot.sol#1202-1595) uses literals with too many digits:
        - _presalePrice = 18000000 * (10 ** 18) (WallStreetJackpot.sol#1255)
WallStreetJackpot.constructor(address) (WallStreetJackpot.sol#2049-2080) uses literals with too many digits:
        - _mint(_marketingWalletAddress,100000000000 * (10 ** 18)) (WallStreetJackpot.sol#2079)
WallStreetJackpot.updateGasForProcessing(uint256) (WallStreetJackpot.sol#2287-2298) uses literals with too many digits:
        - require(bool,string)(newValue >= 200000 && newValue <= 500000,WallStreetJackpot: gasForProcessing must be between 200,000 and 5
00,000) (WallStreetJackpot.sol#2288-2291)
WallStreetJackpot.slitherConstructorVariables() (WallStreetJackpot.sol#1961-2621) uses literals with too many digits:
        - deadWallet = 0x000000000000000000000000000000000000dEaD (WallStreetJackpot.sol#1972)
WallStreetJackpot.slitherConstructorVariables() (WallStreetJackpot.sol#1961-2621) uses literals with too many digits:
        - swapTokensAtAmount = 2000000 * (10 ** 18) (WallStreetJackpot.sol#1976)
WallStreetJackpot.slitherConstructorVariables() (WallStreetJackpot.sol#1961-2621) uses literals with too many digits:
        - gasForProcessing = 300000 (WallStreetJackpot.sol#1988)
```

```
WallStreetJackpotDividendTracker.constructor() (WallStreetJackpot.sol#2647-2656) uses literals with too many digits:
        - minimumTokenBalanceForDividends = 200000 * (10 ** 18) (WallStreetJackpot.sol#2655)
WallStreetJackpotDividendTracker.getAccountAtIndex(uint256) (WallStreetJackpot.sol#2756-2786) uses literals with too many digits:
        - (0x0000000000000000000000000000000000000000,- 1,- 1,0,0,0,0,0) (WallStreetJackpot.sol#2771-2780)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
SafeMathInt.MAX_INT256 (WallStreetJackpot.sol#167) is never used in SafeMathInt (WallStreetJackpot.sol#165-222)
WallStreetJackpot._jackpoting (WallStreetJackpot.sol#1998) is never used in WallStreetJackpot (WallStreetJackpot.sol#1961-2621)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variables
INFO:Detectors:
Jackpot._deadWallet (WallStreetJackpot.sol#1208) should be constant
WallStreetJackpot._jackpoting (WallStreetJackpot.sol#1998) should be constant
WallStreetJackpot._snipeBlockAmount (WallStreetJackpot.sol#2001) should be constant
WallStreetJackpot._sniperProtection (WallStreetJackpot.sol#2003) should be constant
WallStreetJackpot.deadWallet (WallStreetJackpot.sol#1972) should be constant
WallStreetJackpot.swapTokensAtAmount (WallStreetJackpot.sol#1976) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Detectors:
get(IterableMapping.Map,address) should be declared external:
        - IterableMapping.get(IterableMapping.Map,address) (WallStreetJackpot.sol#261-263)
getIndexOfKey(IterableMapping.Map,address) should be declared external:
        - IterableMapping.getIndexOfKey(IterableMapping.Map,address) (WallStreetJackpot.sol#265-274)
getKeyAtIndex(IterableMapping.Map,uint256) should be declared external:
        - IterableMapping.getKeyAtIndex(IterableMapping.Map,uint256) (WallStreetJackpot.sol#276-282)
size(IterableMapping.Map) should be declared external:
        - IterableMapping.size(IterableMapping.Map) (WallStreetJackpot.sol#284-286)
name() should be declared external:
        - ERC20.name() (WallStreetJackpot.sol#838-840)
symbol() should be declared external:
        - ERC20.symbol() (WallStreetJackpot.sol#846-848)
decimals() should be declared external:
        - ERC20.decimals() (WallStreetJackpot.sol#863-865)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (WallStreetJackpot.sol#895-903)
allowance(address,address) should be declared external:
        - ERC20.allowance(address,address) (WallStreetJackpot.sol#908-916)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (WallStreetJackpot.sol#925-933)
```

```
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (WallStreetJackpot.sol#948-963)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (WallStreetJackpot.sol#977-988)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (WallStreetJackpot.sol#1004-1018)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (WallStreetJackpot.sol#1182-1185)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (WallStreetJackpot.sol#1191-1198)
viewtoClaimReward(uint256) should be declared external:
        - Jackpot.viewtoClaimReward(uint256) (WallStreetJackpot.sol#1510-1512)
viewClaimedReward(uint256) should be declared external:
        - Jackpot.viewClaimedReward(uint256) (WallStreetJackpot.sol#1514-1516)
viewTotalTicketsR(uint256) should be declared external:
        - Jackpot.viewTotalTicketsR(uint256) (WallStreetJackpot.sol#1518-1520)
viewWinners(uint256) should be declared external:
        - Jackpot.viewWinners(uint256) (WallStreetJackpot.sol#1522-1528)
viewIsClaimed(uint256,address) should be declared external:
        - Jackpot.viewIsClaimed(uint256,address) (WallStreetJackpot.sol#1530-1536)
viewWinnersReward(uint256,address) should be declared external:
        - Jackpot.viewWinnersReward(uint256,address) (WallStreetJackpot.sol#1538-1544)
viewHoldersTickets(uint256,address) should be declared external:
        - Jackpot.viewHoldersTickets(uint256,address) (WallStreetJackpot.sol#1546-1552)
claimJackpotReward(uint256) should be declared external:
        - Jackpot.claimJackpotReward(uint256) (WallStreetJackpot.sol#1555-1575)
withdrawEth() should be declared external:
        - Jackpot.withdrawEth() (WallStreetJackpot.sol#1577-1584)
withdrawEth(uint256) should be declared external:
        - Jackpot.withdrawEth(uint256) (WallStreetJackpot.sol#1586-1592)
withdrawDividend() should be declared external:
        - DividendPayingToken.withdrawDividend() (WallStreetJackpot.sol#1818-1820)
        - WallStreetJackpotDividendTracker.withdrawDividend() (WallStreetJackpot.sol#2669-2674)
dividendOf(address) should be declared external:
        - DividendPayingToken.dividendOf(address) (WallStreetJackpot.sol#1855-1857)
withdrawnDividendOf(address) should be declared external:
        - DividendPayingToken.withdrawnDividendOf(address) (WallStreetJackpot.sol#1874-1881)
switchJackpot(bool) should be declared external:
```

```
switchJackpot(bool) should be declared external:
        - WallStreetJackpot.switchJackpot(bool) (WallStreetJackpot.sol#2105-2107)
setJackpotPeriod(uint256) should be declared external:
        - WallStreetJackpot.setJackpotPeriod(uint256) (WallStreetJackpot.sol#2109-2111)
setJackpotThreshold(uint256) should be declared external:
        - WallStreetJackpot.setJackpotThreshold(uint256) (WallStreetJackpot.sol#2113-2115)
setTokensAmont(uint256) should be declared external:
        - WallStreetJackpot.setTokensAmont(uint256) (WallStreetJackpot.sol#2117-2119)
enableBasedOnToken(bool) should be declared external:
        - WallStreetJackpot.enableBasedOnToken(bool) (WallStreetJackpot.sol#2121-2123)
setEnableSubTickets(bool) should be declared external:
        - WallStreetJackpot.setEnableSubTickets(bool) (WallStreetJackpot.sol#2125-2127)
excludeJackpot(address) should be declared external:
        - WallStreetJackpot.excludeJackpot(address) (WallStreetJackpot.sol#2129-2131)
setDxsale(address,address) should be declared external:
        - WallStreetJackpot.setDxsale(address,address) (WallStreetJackpot.sol#2133-2138)
setPresalePrice(uint256) should be declared external:
        - WallStreetJackpot.setPresalePrice(uint256) (WallStreetJackpot.sol#2140-2142)
updateJackpot(address) should be declared external:
        - WallStreetJackpot.updateJackpot(address) (WallStreetJackpot.sol#2144-2160)
updateDividendTracker(address) should be declared external:
        - WallStreetJackpot.updateDividendTracker(address) (WallStreetJackpot.sol#2162-2185)
updateUniswapV2Router(address) should be declared external:
        - WallStreetJackpot.updateUniswapV2Router(address) (WallStreetJackpot.sol#2187-2197)
excludeMultipleAccountsFromFees(address[],bool) should be declared external:
        - WallStreetJackpot.excludeMultipleAccountsFromFees(address[],bool) (WallStreetJackpot.sol#2209-2218)
setAutomatedMarketMakerPair(address,bool) should be declared external:
```



static analysis

WallStreetJackpot.sol

Transaction origin:

Use of tx.origin: "tx.origin" is useful only in very exceptional cases.
If you use it for authentication, you usually want to replace it by "msg.sender", because otherwise any contract you call can act on your behalf.
more
Pos: 435:12:

Transaction origin:

Use of tx.origin: "tx.origin" is useful only in very exceptional cases.
If you use it for authentication, you usually want to replace it by "msg.sender", because otherwise any contract you call can act on your behalf.
more
Pos: 566:20:

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in DividendPayingToken._withdrawDividendOfUser(address payable): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 95:4:

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in Jackpot.claimJackpotReward(uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 363:4:

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in Jackpot.withdrawEth(): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 385:4:

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in Jackpot.withdrawEth(uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 394:4:

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in WallStreetJackpot.(address): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 102:4:

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in WallStreetJackpot.updateJackpot(address): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 197:4:

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in WallStreetJackpot.updateDividendTracker(address): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 215:4:

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in WallStreetJackpot.updateUniswapV2Router(address): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 240:4:

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in WallStreetJackpot._setAutomatedMarketMakerPair(address,bool): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 326:4:

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in WallStreetJackpot.processDividendTracker(uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 423:4:

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in WallStreetJackpot._transfer(address,address,uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 451:4:

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in WallStreetJackpot.setDxSaleAddress(address,address): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 572:4:

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in WallStreetJackpot.swapTokensForEth(uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 631:4:

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in WallStreetJackpot.swapAndSendDividends(uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 664:4:

Block timestamp:

Use of "now": "now" does not mean current time. "now" is an alias for "block.timestamp".
"block.timestamp" can be influenced by miners to a certain degree, be careful.

more

Pos: 312:43:

Block timestamp:
Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree.
That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
more
Pos: 164:13:

Block timestamp:
Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree.
That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
more
Pos: 172:29:

Block timestamp:
Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree.
That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
more
Pos: 589:33:

Block timestamp:
Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree.
That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
more
Pos: 645:12:

Block timestamp:
Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree.
That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
more
Pos: 660:12:

Block timestamp:
Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree.
That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
more
Pos: 804:57:

Block timestamp:
Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree.
That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
more
Pos: 805:32:

Block timestamp:
Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree.
That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
more
Pos: 842:28:

Block timestamp:
Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree.
That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
more
Pos: 846:15:

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree.
That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
more
Pos: 930:38:

Low level calls:

Use of "call": should be avoided whenever possible.
It can lead to unexpected behavior if return value is not handled properly.
Please use Direct Calls via specifying the called contract's interface.
more
Pos: 105:31:

Low level calls:

Use of "call": should be avoided whenever possible.
It can lead to unexpected behavior if return value is not handled properly.
Please use Direct Calls via specifying the called contract's interface.
more
Pos: 373:27:

Low level calls:

Use of "call": should be avoided whenever possible.
It can lead to unexpected behavior if return value is not handled properly.
Please use Direct Calls via specifying the called contract's interface.
more
Pos: 388:27:

Low level calls:

Use of "call": should be avoided whenever possible.
It can lead to unexpected behavior if return value is not handled properly.
Please use Direct Calls via specifying the called contract's interface.
more
Pos: 396:27:

Low level calls:

Use of "call": should be avoided whenever possible.
It can lead to unexpected behavior if return value is not handled properly.
Please use Direct Calls via specifying the called contract's interface.
more
Pos: 668:27:

Gas & Economy

Gas costs:

Gas requirement of function DividendPayingToken.distributeDividends is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 72:4:

Gas costs:

Gas requirement of function WallStreetJackpotDividendTracker.distributeDividends is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 72:4:

Gas costs:

Gas requirement of function DividendPayingToken.withdrawDividend is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 89:4:

Gas costs:

Gas requirement of function DividendPayingToken.dividendOf is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 126:4:

Gas costs:

Gas requirement of function WallStreetJackpotDividendTracker.dividendOf is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 126:4:

Gas costs:

Gas requirement of function DividendPayingToken.withdrawableDividendOf is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 133:4:

Gas costs:

Gas requirement of function WallStreetJackpot.withdrawableDividendOf is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 133:4:

Gas costs:

Gas requirement of function WallStreetJackpotDividendTracker.withdrawableDividendOf is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 133:4:

Gas costs:

Gas requirement of function DividendPayingToken.accumulativeDividendOf is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 159:4:

Gas costs:

Gas requirement of function WallStreetJackpotDividendTracker.accumulativeDividendOf is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 159:4:

Gas costs:

Gas requirement of function DividendPayingToken.name is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 63:4:

Gas costs:

Gas requirement of function ERC20.name is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 63:4:

Gas costs:

Gas requirement of function DividendPayingToken.symbol is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 71:4:

Gas costs:

Gas requirement of function ERC20.symbol is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 71:4:

Gas costs:

Gas requirement of function WallStreetJackpot.symbol is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 71:4:

Gas costs:

Gas requirement of function WallStreetJackpotDividendTracker.symbol is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 71:4:

Gas costs:

Gas requirement of function ERC20.transfer is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 120:4:

Gas costs:

Gas requirement of function DividendPayingToken.approve is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 150:4:

Gas costs:

Gas requirement of function ERC20.approve is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 150:4:

Gas costs:

Gas requirement of function WallStreetJackpot.approve is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 150:4:

Gas costs:

Gas requirement of function WallStreetJackpotDividendTracker.approve is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 150:4:

Gas costs:

Gas requirement of function ERC20.transferFrom is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 173:4:

Gas costs:

Gas requirement of function WallStreetJackpot.transferFrom is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 173:4:

Gas costs:

Gas requirement of function WallStreetJackpotDividendTracker.transferFrom is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 173:4:

Gas costs:

Gas requirement of function DividendPayingToken.increaseAllowance is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 202:4:

Gas costs:

Gas requirement of function ERC20.increaseAllowance is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 202:4:

Gas costs:

Gas requirement of function WallStreetJackpot.increaseAllowance is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 202:4:

Gas costs:

Gas requirement of function WallStreetJackpotDividendTracker.increaseAllowance is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 202:4:

Gas costs:

Gas requirement of function DividendPayingToken.decreaseAllowance is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 229:4:

Gas costs:

Gas requirement of function ERC20.decreaseAllowance is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 229:4:

Gas costs:

Gas requirement of function WallStreetJackpot.setTokensAmont is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 73:4:

Gas costs:

Gas requirement of function WallStreetJackpot.setJackpotThreshold is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 81:4:

Gas costs:

Gas requirement of function WallStreetJackpot.setJackpotPeriod is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 85:4:

Gas costs:

Gas requirement of function WallStreetJackpot.enableBasedOnToken is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 89:4:

Gas costs:

Gas requirement of function WallStreetJackpot.excludeJackpot is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 93:4:

Gas costs:

Gas requirement of function WallStreetJackpot.setPresalePrice is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 97:4:

Gas costs:

Gas requirement of function WallStreetJackpot.setDxsale is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 101:4:

Gas costs:

Gas requirement of function WallStreetJackpot.setPrizePercent is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 111:4:

Gas costs:

Gas requirement of function WallStreetJackpot.setPrizeOdds is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 123:4:

Gas costs:

Gas requirement of function WallStreetJackpot.setPrize is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 133:4:

Gas costs:

Gas requirement of function WallStreetJackpot.setEnableSubTickets is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 138:4:

Gas costs:

Gas requirement of function Jackpot.setMarketWallet is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 142:4:

Gas costs:

Gas requirement of function WallStreetJackpot.setMarketWallet is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 142:4:

Gas costs:

Gas requirement of function Jackpot.jackpot is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 150:4:

Gas costs:

Gas requirement of function Jackpot.viewWinners is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 330:4:

Gas costs:

Gas requirement of function Jackpot.claimJackpotReward is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 363:4:

Gas costs:

Gas requirement of function Jackpot.withdrawEth is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 385:4:

Gas costs:

Gas requirement of function Jackpot.withdrawEth is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 394:4:

Gas costs:

Gas requirement of function Jackpot.transferOwnership is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 55:4:

Gas costs:

Gas requirement of function Ownable.transferOwnership is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 55:4:

Gas costs:

Gas requirement of function WallStreetJackpot.transferOwnership is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 55:4:

Gas costs:

Gas requirement of function WallStreetJackpotDividendTracker.transferOwnership is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 55:4:

Gas costs:

Gas requirement of function WallStreetJackpot.setPrize is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 137:4:

Gas costs:

Gas requirement of function WallStreetJackpot.setPrizePercent is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 141:4:

Gas costs:

Gas requirement of function WallStreetJackpot.setPrizeOdds is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 150:4:

Gas costs:

Gas requirement of function WallStreetJackpot.setJackpotPeriod is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 162:4:

Gas costs:

Gas requirement of function WallStreetJackpot.setJackpotThreshold is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 166:4:

Gas costs:

Gas requirement of function WallStreetJackpot.setTokensAmont is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 170:4:

Gas costs:

Gas requirement of function WallStreetJackpot.enableBasedOnToken is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 174:4:

Gas costs:

Gas requirement of function WallStreetJackpot.setEnableSubTickets is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 178:4:

Gas costs:

Gas requirement of function WallStreetJackpot.excludeJackpot is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 182:4:

Gas costs:

Gas requirement of function WallStreetJackpot.setDxsale is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 186:4:

Gas costs:

Gas requirement of function WallStreetJackpot.setPresalePrice is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 193:4:

Gas costs:

Gas requirement of function WallStreetJackpot.updateJackpot is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 197:4:

Gas costs:

Gas requirement of function WallStreetJackpot.updateDividendTracker is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 215:4:

Gas costs:

Gas requirement of function WallStreetJackpot.updateUniswapV2Router is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 240:4:

Gas costs:

Gas requirement of function WallStreetJackpot.excludeFromFees is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 252:4:

Gas costs:

Gas requirement of function WallStreetJackpot.excludeMultipleAccountsFromFees is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 262:4:

Gas costs:

Gas requirement of function Jackpot.setMarketWallet is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 273:4:

Gas costs:

Gas requirement of function WallStreetJackpot.setMarketWallet is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 273:4:

Gas costs:

Gas requirement of function WallStreetJackpot.setBNBRewardsFee is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 282:4:

Gas costs:

Gas requirement of function WallStreetJackpot.setLiquiditFee is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 289:4:

Gas costs:

Gas requirement of function WallStreetJackpot.setMarketingFee is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 296:4:

Gas costs:

Gas requirement of function WallStreetJackpot.setJackpotFee is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 303:4:

Gas costs:

Gas requirement of function WallStreetJackpot.setAutomatedMarketMakerPair is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 310:4:

Gas costs:

Gas requirement of function WallStreetJackpot.updateGasForProcessing is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 340:4:

Gas costs:

Gas requirement of function WallStreetJackpot.updateClaimWait is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 353:4:

Gas costs:

Gas requirement of function WallStreetJackpotDividendTracker.updateClaimWait is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 353:4:

Gas costs:

Gas requirement of function WallStreetJackpot.getClaimWait is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 357:4:

Gas costs:

Gas requirement of function WallStreetJackpot.getTotalDividendsDistributed is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 361:4:

Gas costs:

Gas requirement of function DividendPayingToken.withdrawableDividendOf is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 369:4:

Gas costs:

Gas requirement of function WallStreetJackpot.withdrawableDividendOf is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 369:4:

Gas costs:

Gas requirement of function WallStreetJackpotDividendTracker.withdrawableDividendOf is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 369:4:

Gas costs:

Gas requirement of function WallStreetJackpot.dividendTokenBalanceOf is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 377:4:

Gas costs:

Gas requirement of function WallStreetJackpot.excludeFromDividends is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 385:4:

Gas costs:

Gas requirement of function WallStreetJackpotDividendTracker.excludeFromDividends is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 385:4:

Gas costs:

Gas requirement of function WallStreetJackpot.getAccountDividendsInfo is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 389:4:

Gas costs:

Gas requirement of function WallStreetJackpot.getAccountDividendsInfoAtIndex is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 406:4:

Gas costs:

Gas requirement of function WallStreetJackpot.processDividendTracker is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 423:4:

Gas costs:

Gas requirement of function WallStreetJackpot.claim is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 439:4:

Gas costs:

Gas requirement of function WallStreetJackpot.getLastProcessedIndex is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 443:4:

Gas costs:

Gas requirement of function WallStreetJackpot.getNumberOfDividendTokenHolders is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 447:4:

Gas costs:

Gas requirement of function WallStreetJackpot.setDxSaleAddress is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 572:4:

Gas costs:

Gas requirement of function WallStreetJackpot.enableTrading is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 585:4:

Gas costs:

Gas requirement of function DividendPayingToken.withdrawDividend is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 722:4:

Gas costs:

Gas requirement of function WallStreetJackpotDividendTracker.withdrawDividend is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 722:4:

Gas costs:

Gas requirement of function WallStreetJackpot.excludeFromDividends is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 729:4:

Gas costs:

Gas requirement of function WallStreetJackpotDividendTracker.excludeFromDividends is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 729:4:

Gas costs:

Gas requirement of function WallStreetJackpot.updateClaimWait is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 739:4:

Gas costs:

Gas requirement of function WallStreetJackpotDividendTracker.updateClaimWait is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 739:4:

Gas costs:

Gas requirement of function WallStreetJackpot.getLastProcessedIndex is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 752:4:

Gas costs:

Gas requirement of function WallStreetJackpotDividendTracker.getAccount is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 760:4:

Gas costs:

Gas requirement of function WallStreetJackpotDividendTracker.getAccountAtIndex is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 809:4:

Gas costs:

Gas requirement of function WallStreetJackpotDividendTracker.setBalance is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 849:4:

Gas costs:

Gas requirement of function WallStreetJackpotDividendTracker.process is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 868:4:

Gas costs:

Gas requirement of function WallStreetJackpotDividendTracker.processAccount is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 922:4:

For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point.
Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.
more
Pos: 266:8:

For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point.
Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.
more
Pos: 894:12:

ERC

ERC20:

ERC20 contract's "decimals" function should have "uint8" as return type
more
Pos: 17:4:

Miscellaneous

Constant/View/Pure functions:

ERC20._beforeTokenTransfer(address,address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 359:4:

Constant/View/Pure functions:

IterableMapping.set(struct IterableMapping.Map,address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 40:4:

Constant/View/Pure functions:

Jackpot.getTokenPerBNBPrice(address,uint256) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 291:4:

Constant/View/Pure functions:

Jackpot.random(uint256) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 309:4:

Constant/View/Pure functions:

SafeMath.sub(uint256,uint256) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 33:4:

Constant/View/Pure functions:

SafeMath.div(uint256,uint256) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 94:4:

Constant/View/Pure functions:

SafeMath.mod(uint256,uint256) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 134:4:

Constant/View/Pure functions:

WallStreetJackpot.getClaimWait() : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 357:4:

Constant/View/Pure functions:

WallStreetJackpot.getTotalDividendsDistributed() : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 361:4:

Constant/View/Pure functions:

WallStreetJackpot.getAccountDividendsInfo(address) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 389:4:

Constant/View/Pure functions:

WallStreetJackpot.getAccountDividendsInfoAtIndex(uint256) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 406:4:

Constant/View/Pure functions:

WallStreetJackpotDividendTracker._transfer(address,address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 711:4:

Constant/View/Pure functions:

WallStreetJackpotDividendTracker.withdrawDividend() : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 722:4:

Constant/View/Pure functions:

WallStreetJackpotDividendTracker.getAccount(address) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 760:4:

Constant/View/Pure functions:

WallStreetJackpotDividendTracker.getAccountAtIndex(uint256) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 809:4:

Similar variable names:

ERC20.(string,string) : Variables have very similar names "_name" and "name_". Note: Modifiers are currently not considered by this static analysis.
Pos: 56:8:

Similar variable names:

ERC20.(string,string) : Variables have very similar names "_name" and "name_". Note: Modifiers are currently not considered by this static analysis.
Pos: 56:16:

Similar variable names:

ERC20.(string,string) : Variables have very similar names "_symbol" and "symbol_". Note: Modifiers are currently not considered by this static analysis.
Pos: 57:8:

Similar variable names:

ERC20.(string,string) : Variables have very similar names "_symbol" and "symbol_". Note: Modifiers are currently not considered by this static analysis.
Pos: 57:18:

Similar variable names:

ERC20._mint(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 287:16:

Similar variable names:

ERC20._mint(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 289:41:

Similar variable names:

ERC20._mint(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 289:50:

Similar variable names:

ERC20._mint(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 291:40:

Similar variable names:

ERC20._mint(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 292:18:

Similar variable names:

ERC20._mint(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 292:39:

Similar variable names:

ERC20._mint(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 292:52:

Similar variable names:

ERC20._mint(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 293:34:

Similar variable names:

ERC20._mint(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 293:43:

Similar variable names:

ERC20._burn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 308:16:

Similar variable names:

ERC20._burn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 310:29:

Similar variable names:

ERC20._burn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 310:50:

Similar variable names:

ERC20._burn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 312:18:

Similar variable names:

ERC20._burn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 312:39:

Similar variable names:

ERC20._burn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 313:12:

Similar variable names:

ERC20._burn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 316:40:

Similar variable names:

ERC20._burn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 317:22:

Similar variable names:

ERC20._burn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 317:43:

Similar variable names:

IterableMapping.set(struct IterableMapping.Map,address,uint256) : Variables have very similar names "map" and "val". Note: Modifiers are currently not considered by this static analysis.
Pos: 45:12:

Similar variable names:

IterableMapping.set(struct IterableMapping.Map,address,uint256) : Variables have very similar names "map" and "val". Note: Modifiers are currently not considered by this static analysis.
Pos: 46:12:

Similar variable names:

IterableMapping.set(struct IterableMapping.Map,address,uint256) : Variables have very similar names "map" and "val". Note: Modifiers are currently not considered by this static analysis.
Pos: 46:30:

Similar variable names:

IterableMapping.set(struct IterableMapping.Map,address,uint256) : Variables have very similar names "map" and "val". Note: Modifiers are currently not considered by this static analysis.
Pos: 48:12:

Similar variable names:

IterableMapping.set(struct IterableMapping.Map,address,uint256) : Variables have very similar names "map" and "val". Note: Modifiers are currently not considered by this static analysis.
Pos: 49:12:

Similar variable names:

IterableMapping.set(struct IterableMapping.Map,address,uint256) : Variables have very similar names "map" and "val". Note: Modifiers are currently not considered by this static analysis.
Pos: 49:30:

Similar variable names:

IterableMapping.set(struct IterableMapping.Map,address,uint256) : Variables have very similar names "map" and "val". Note: Modifiers are currently not considered by this static analysis.
Pos: 50:12:

Similar variable names:

IterableMapping.set(struct IterableMapping.Map,address,uint256) : Variables have very similar names "map" and "val". Note: Modifiers are currently not considered by this static analysis.
Pos: 50:31:

Similar variable names:

IterableMapping.set(struct IterableMapping.Map,address,uint256) : Variables have very similar names "map" and "val". Note: Modifiers are currently not considered by this static analysis.
Pos: 51:12:

Similar variable names:

Jackpot.excludeJackpot(address) : Variables have very similar names "_holders" and "holder". Note: Modifiers are currently not considered by this static analysis.
Pos: 94:17:

Similar variable names:

Jackpot.setPrize(uint256,uint256) : Variables have very similar names "_minPrize" and "_maxPrize". Note: Modifiers are currently not considered by this static analysis.
Pos: 134:8:

Similar variable names:

Jackpot.setPrize(uint256,uint256) : Variables have very similar names "_minPrize" and "_maxPrize". Note: Modifiers are currently not considered by this static analysis.
Pos: 135:8:

Similar variable names:

Jackpot.setPrize(uint256,uint256) : Variables have very similar names "minPrize" and "maxPrize". Note: Modifiers are currently not considered by this static analysis.
Pos: 134:20:

Similar variable names:

Jackpot.setPrize(uint256,uint256) : Variables have very similar names "minPrize" and "maxPrize". Note: Modifiers are currently not considered by this static analysis.
Pos: 135:20:

Similar variable names:

Jackpot.jackpot(address,address,address,uint256) : Variables have very similar names "_minPrize" and "_maxPrize". Note: Modifiers are currently not considered by this static analysis.
Pos: 159:36:

Similar variable names:

Jackpot.jackpot(address,address,address,uint256) : Variables have very similar names "_minPrize" and "_maxPrize". Note: Modifiers are currently not considered by this static analysis.
Pos: 159:48:

Similar variable names:

Jackpot.jackpot(address,address,address,uint256) : Variables have very similar names "_minPrize" and "_maxPrize". Note: Modifiers are currently not considered by this static analysis.
Pos: 162:27:

Similar variable names:

Jackpot.jackpot(uint256,uint256) : Variables have very similar names "_totalTickets" and "_totalTicketsR". Note: Modifiers are currently not considered by this static analysis.
Pos: 201:31:

Similar variable names:

Jackpot.jackpot(uint256,uint256) : Variables have very similar names "_winners" and "winner1". Note: Modifiers are currently not considered by this static analysis.
Pos: 203:13:

Similar variable names:

Jackpot.jackpot(uint256,uint256) : Variables have very similar names "_winners" and "winner1". Note: Modifiers are currently not considered by this static analysis.
Pos: 221:12:

Similar variable names:

Jackpot.jackpot(uint256,uint256) : Variables have very similar names "_winners" and "winner1". Note: Modifiers are currently not considered by this static analysis.
Pos: 221:31:

Similar variable names:

Jackpot.jackpot(uint256,uint256) : Variables have very similar names "_winners" and "winner2". Note: Modifiers are currently not considered by this static analysis.
Pos: 209:13:

Similar variable names:

Jackpot.jackpot(uint256,uint256) : Variables have very similar names "_winners" and "winner2". Note: Modifiers are currently not considered by this static analysis.
Pos: 221:12:

Similar variable names:

Jackpot.jackpot(uint256,uint256) : Variables have very similar names "_winners" and "winner2". Note: Modifiers are currently not considered by this static analysis.
Pos: 221:40:

Similar variable names:

Jackpot.jackpot(uint256,uint256) : Variables have very similar names "_winners" and "winner3". Note: Modifiers are currently not considered by this static analysis.
Pos: 215:13:

Similar variable names:

Jackpot.jackpot(uint256,uint256) : Variables have very similar names "_winners" and "winner3". Note: Modifiers are currently not considered by this static analysis.
Pos: 221:12:

Similar variable names:

Jackpot.jackpot(uint256,uint256) : Variables have very similar names "_winners" and "winner3". Note: Modifiers are currently not considered by this static analysis.
Pos: 221:49:

Similar variable names:

Jackpot.jackpot(uint256,uint256) : Variables have very similar names "_totalTicketsR" and "totalTickets". Note: Modifiers are currently not considered by this static analysis.
Pos: 201:8:

Similar variable names:

Jackpot.jackpot(uint256,uint256) : Variables have very similar names "_totalTicketsR" and "totalTickets". Note: Modifiers are currently not considered by this static analysis.
Pos: 202:12:

Similar variable names:

Jackpot.jackpot(uint256,uint256) : Variables have very similar names "_totalTicketsR" and "totalTickets". Note: Modifiers are currently not considered by this static analysis.
Pos: 206:16:

Similar variable names:

Jackpot.jackpot(uint256,uint256) : Variables have very similar names "_totalTicketsR" and "totalTickets". Note: Modifiers are currently not considered by this static analysis.
Pos: 212:16:

Similar variable names:

Jackpot.jackpot(uint256,uint256) : Variables have very similar names "_totalTicketsR" and "totalTickets". Note: Modifiers are currently not considered by this static analysis.
Pos: 218:16:

Similar variable names:

Jackpot.jackpot(uint256,uint256) : Variables have very similar names "amount" and "amount1". Note: Modifiers are currently not considered by this static analysis.
Pos: 203:30:

Similar variable names:

Jackpot.jackpot(uint256,uint256) : Variables have very similar names "amount" and "amount1". Note: Modifiers are currently not considered by this static analysis.
Pos: 223:12:

Similar variable names:

Jackpot.jackpot(uint256,uint256) : Variables have very similar names "amount" and "amount1". Note: Modifiers are currently not considered by this static analysis.
Pos: 223:29:

Similar variable names:

Jackpot.jackpot(uint256,uint256) : Variables have very similar names "amount" and "amount1". Note: Modifiers are currently not considered by this static analysis.
Pos: 224:58:

Similar variable names:

Jackpot.jackpot(uint256,uint256) : Variables have very similar names "amount" and "amount1". Note: Modifiers are currently not considered by this static analysis.
Pos: 225:36:

Similar variable names:

Jackpot.jackpot(uint256,uint256) : Variables have very similar names "amount" and "amount2". Note: Modifiers are currently not considered by this static analysis.
Pos: 209:30:

Similar variable names:

Jackpot.jackpot(uint256,uint256) : Variables have very similar names "amount" and "amount2". Note: Modifiers are currently not considered by this static analysis.
Pos: 223:12:

Similar variable names:

Jackpot.jackpot(uint256,uint256) : Variables have very similar names "amount" and "amount2". Note: Modifiers are currently not considered by this static analysis.
Pos: 223:41:

Similar variable names:

Jackpot.jackpot(uint256,uint256) : Variables have very similar names "amount" and "amount2". Note: Modifiers are currently not considered by this static analysis.
Pos: 224:58:

Similar variable names:

Jackpot.jackpot(uint256,uint256) : Variables have very similar names "amount" and "amount2". Note: Modifiers are currently not considered by this static analysis.
Pos: 225:36:

Similar variable names:

Jackpot.jackpot(uint256,uint256) : Variables have very similar names "amount" and "amount3". Note: Modifiers are currently not considered by this static analysis.
Pos: 215:30:

Similar variable names:

Jackpot.jackpot(uint256,uint256) : Variables have very similar names "amount" and "amount3". Note: Modifiers are currently not considered by this static analysis.
Pos: 223:12:

Similar variable names:

Jackpot.jackpot(uint256,uint256) : Variables have very similar names "amount" and "amount3". Note: Modifiers are currently not considered by this static analysis.
Pos: 223:54:

Similar variable names:

Jackpot.jackpot(uint256,uint256) : Variables have very similar names "amount" and "amount3". Note: Modifiers are currently not considered by this static analysis.
Pos: 224:58:

Similar variable names:

Jackpot.jackpot(uint256,uint256) : Variables have very similar names "amount" and "amount3". Note: Modifiers are currently not considered by this static analysis.
Pos: 225:36:

Similar variable names:

Jackpot.addTickets(uint256,address,uint256) : Variables have very similar names "_totalTickets" and "_totalTicketsR". Note: Modifiers are currently not considered by this static analysis.
Pos: 265:12:

Similar variable names:

Jackpot.addTickets(uint256,address,uint256) : Variables have very similar names "_totalTickets" and "_totalTicketsR". Note: Modifiers are currently not considered by this static analysis.
Pos: 265:36:

Similar variable names:

Jackpot.addTickets(uint256,address,uint256) : Variables have very similar names "_totalTickets" and "_totalTicketsR". Note: Modifiers are currently not considered by this static analysis.
Pos: 266:12:

Similar variable names:

Jackpot.addTickets(uint256,address,uint256) : Variables have very similar names "_holders" and "holder". Note: Modifiers are currently not considered by this static analysis.
Pos: 261:37:

Similar variable names:

Jackpot.addTickets(uint256,address,uint256) : Variables have very similar names "_holders" and "holder". Note: Modifiers are currently not considered by this static analysis.
Pos: 262:35:

Similar variable names:

Jackpot.addTickets(uint256,address,uint256) : Variables have very similar names "_holders" and "holder". Note: Modifiers are currently not considered by this static analysis.
Pos: 268:33:

Similar variable names:

Jackpot.addTickets(uint256,address,uint256) : Variables have very similar names "_holders" and "holder". Note: Modifiers are currently not considered by this static analysis.
Pos: 269:32:

Similar variable names:

Jackpot.addTickets(uint256,address,uint256) : Variables have very similar names "_holders" and "holder". Note: Modifiers are currently not considered by this static analysis.
Pos: 270:16:

Similar variable names:

Jackpot.addTickets(uint256,address,uint256) : Variables have very similar names "_holders" and "holder". Note: Modifiers are currently not considered by this static analysis.
Pos: 270:37:

Similar variable names:

Jackpot.subTickets(uint256,address,uint256) : Variables have very similar names "_totalTickets" and "_totalTicketsR". Note: Modifiers are currently not considered by this static analysis.
Pos: 285:12:

Similar variable names:

Jackpot.subTickets(uint256,address,uint256) : Variables have very similar names "_totalTickets" and "_totalTicketsR". Note: Modifiers are currently not considered by this static analysis.
Pos: 286:12:

Similar variable names:

Jackpot.subTickets(uint256,address,uint256) : Variables have very similar names "_holders" and "holder". Note: Modifiers are currently not considered by this static analysis.
Pos: 281:37:

Similar variable names:

Jackpot.subTickets(uint256,address,uint256) : Variables have very similar names "_holders" and "holder". Note: Modifiers are currently not considered by this static analysis.
Pos: 282:35:

Similar variable names:

Jackpot.subTickets(uint256,address,uint256) : Variables have very similar names "_holders" and "holder". Note: Modifiers are currently not considered by this static analysis.
Pos: 282:68:

Similar variable names:

Jackpot.viewTotalTicketsR(uint256) : Variables have very similar names "_totalTickets" and "_totalTicketsR". Note: Modifiers are currently not considered by this static analysis.
Pos: 327:15:

Similar variable names:

Jackpot.viewIsClaimed(uint256,address) : Variables have very similar names "_holders" and "holder". Note: Modifiers are currently not considered by this static analysis.
Pos: 343:33:

Similar variable names:

Jackpot.viewWinnersReward(uint256,address) : Variables have very similar names "_holders" and "holder". Note: Modifiers are currently not considered by this static analysis.
Pos: 351:37:

Similar variable names:

Jackpot.viewHoldersTickets(uint256,address) : Variables have very similar names "_holders" and "holder". Note: Modifiers are currently not considered by this static analysis.
Pos: 359:38:

Similar variable names:

SafeMathInt.mul(int256,int256) : Variables have very similar names "MIN_INT256" and "MAX_INT256". Note: Modifiers are currently not considered by this static analysis.
Pos: 45:21:

Similar variable names:

SafeMathInt.mul(int256,int256) : Variables have very similar names "MIN_INT256" and "MAX_INT256". Note: Modifiers are currently not considered by this static analysis.
Pos: 45:40:

Similar variable names:

SafeMathInt.mul(int256,int256) : Variables have very similar names "MIN_INT256" and "MAX_INT256". Note: Modifiers are currently not considered by this static analysis.
Pos: 45:60:

Similar variable names:

SafeMathInt.div(int256,int256) : Variables have very similar names "MIN_INT256" and "MAX_INT256". Note: Modifiers are currently not considered by this static analysis.
Pos: 55:32:

Similar variable names:

SafeMathInt.abs(int256) : Variables have very similar names "MIN_INT256" and "MAX_INT256". Note: Modifiers are currently not considered by this static analysis.
Pos: 83:21:

Similar variable names:

WallStreetJackpot.setPrize(uint256,uint256) : Variables have very similar names "minPrize" and "maxPrize". Note: Modifiers are currently not considered by this static analysis.
Pos: 138:25:

Similar variable names:

WallStreetJackpot.setPrize(uint256,uint256) : Variables have very similar names "minPrize" and "maxPrize". Note: Modifiers are currently not considered by this static analysis.
Pos: 138:35:

Similar variable names:

WallStreetJackpotDividendTracker.getAccount(address) : Variables have very similar names "lastClaimTime" and "lastClaimTimes". Note: Modifiers are currently not considered by this static analysis.
Pos: 800:8:

Similar variable names:

WallStreetJackpotDividendTracker.getAccount(address) : Variables have very similar names "lastClaimTime" and "lastClaimTimes". Note: Modifiers are currently not considered by this static analysis.
Pos: 800:24:

Similar variable names:

WallStreetJackpotDividendTracker.getAccount(address) : Variables have very similar names "lastClaimTime" and "lastClaimTimes". Note: Modifiers are currently not considered by this static analysis.
Pos: 802:24:

Similar variable names:

WallStreetJackpotDividendTracker.getAccount(address) : Variables have very similar names "lastClaimTime" and "lastClaimTimes". Note: Modifiers are currently not considered by this static analysis.
Pos: 802:44:

Similar variable names:

WallStreetJackpotDividendTracker.canAutoClaim(uint256) : Variables have very similar names "lastClaimTime" and "lastClaimTimes". Note: Modifiers are currently not considered by this static analysis.
Pos: 842:12:

Similar variable names:

WallStreetJackpotDividendTracker.canAutoClaim(uint256) : Variables have very similar names "lastClaimTime" and "lastClaimTimes". Note: Modifiers are currently not considered by this static analysis.
Pos: 846:35:

Similar variable names:

WallStreetJackpotDividendTracker.processAccount(address payable,bool) : Variables have very similar names "amount" and "account". Note: Modifiers are currently not considered by this static analysis.
Pos: 927:8:

Similar variable names:

WallStreetJackpotDividendTracker.processAccount(address payable,bool) : Variables have very similar names "amount" and "account". Note: Modifiers are currently not considered by this static analysis.
Pos: 927:49:

Similar variable names:

WallStreetJackpotDividendTracker.processAccount(address payable,bool) : Variables have very similar names "amount" and "account". Note: Modifiers are currently not considered by this static analysis.
Pos: 929:12:

Similar variable names:

WallStreetJackpotDividendTracker.processAccount(address payable,bool) : Variables have very similar names "amount" and "account". Note: Modifiers are currently not considered by this static analysis.
Pos: 930:27:

Similar variable names:

WallStreetJackpotDividendTracker.processAccount(address payable,bool) : Variables have very similar names "amount" and "account". Note: Modifiers are currently not considered by this static analysis.
Pos: 931:23:

Similar variable names:

WallStreetJackpotDividendTracker.processAccount(address payable,bool) : Variables have very similar names "amount" and "account". Note: Modifiers are currently not considered by this static analysis.
Pos: 931:32:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 73:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 183:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 264:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 265:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 287:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 308:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 338:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 339:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 41:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 53:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 67:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 87:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 143:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 365:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 386:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 395:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 35:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 56:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 18:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 52:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 77:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 115:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 155:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 45:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 46:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 55:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 66:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 75:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 83:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 88:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 12:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 198:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 205:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 216:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 225:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 241:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 253:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 274:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 314:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 327:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 341:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 345:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 456:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 457:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 458:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 477:16:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 472:16:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 485:12:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 716:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 723:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 730:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 740:8:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 744:8:

Delete from dynamic array:

Using "delete" on an array leaves a gap. The length of the array remains the same. If you want to remove the empty position you need to shift items manually and update the "length" property.
more
Pos: 60:8:

Delete from dynamic array:

Using "delete" on an array leaves a gap. The length of the array remains the same. If you want to remove the empty position you need to shift items manually and update the "length" property.
more
Pos: 61:8:

Delete from dynamic array:

Using "delete" on an array leaves a gap. The length of the array remains the same. If you want to remove the empty position you need to shift items manually and update the "length" property.
more
Pos: 68:8:

Data truncated:

Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 9:24:

Data truncated:

Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 12:20:

Data truncated:

Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 12:21:

Data truncated:

Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 77:16:

Data truncated:

Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 166:12:

Data truncated:

Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 42:25:

Data truncated:

Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 54:22:

Data truncated:

Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 68:25:

Data truncated:

Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 88:33:

Data truncated:

Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 77:16:

Data truncated:

Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 116:20:

Data truncated:

Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 46:29:

Data truncated:

Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 58:15:

Solhint Linter

WallStreetJackpot.sol

```
WallStreetJackpot.sol:3:1: Error: Compiler version ^0.6.2 does not
satisfy the r semver requirement

WallStreetJackpot.sol:14:1: Error: Contract has 23 states
declarations but allowed no more than 15

WallStreetJackpot.sol:33:20: Error: Variable name must be in
mixedCase

WallStreetJackpot.sol:135:32: Error: Code contains empty blocks

WallStreetJackpot.sol:435:13: Error: Avoid to use tx.origin

WallStreetJackpot.sol:549:9: Error: Code contains empty blocks

WallStreetJackpot.sol:549:18: Error: Code contains empty blocks

WallStreetJackpot.sol:550:68: Error: Code contains empty blocks

WallStreetJackpot.sol:550:77: Error: Code contains empty blocks

WallStreetJackpot.sol:566:21: Error: Avoid to use tx.origin

WallStreetJackpot.sol:568:21: Error: Code contains empty blocks

WallStreetJackpot.sol:589:34: Error: Avoid to make time-based
decisions in your business logic

WallStreetJackpot.sol:645:13: Error: Avoid to make time-based
decisions in your business logic

WallStreetJackpot.sol:660:13: Error: Avoid to make time-based
decisions in your business logic

WallStreetJackpot.sol:668:28: Error: Avoid to use low level calls.

WallStreetJackpot.sol:804:58: Error: Avoid to make time-based
decisions in your business logic

WallStreetJackpot.sol:805:33: Error: Avoid to make time-based
decisions in your business logic

WallStreetJackpot.sol:842:29: Error: Avoid to make time-based
decisions in your business logic

WallStreetJackpot.sol:846:16: Error: Avoid to make time-based
decisions in your business logic
```

```
WallStreetJackpot.sol:930:39: Error: Avoid to make time-based
decisions in your business logic
```

Software analysis result:

These software reported many false positive results and some are informational issues. So, those issues can be safely ignored.

Email: audit@EtherAuthority.io