Cyberscope

Audit Report

OnedollarwifHat

May 2024

Network BSC

Address 0xbe896a6ebd3d3ea465c8c34168483ee940431360

Audited by © cyberscope

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	DDP	Decimal Division Precision	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	PAV	Pair Address Validation	Unresolved
●	RED	Redudant Event Declaration	Unresolved
●	RCS	Redundant Conditional Statement	Unresolved
●	RIC	Redundant If Condition	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	RSD	Redundant Swap Duplication	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved

Table of Contents

Review

Contract Name	ODWIF
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	200 runs
Explorer	https://bscscan.com/address/0xbe896a6ebd3d3ea465c8c34168483ee94043136a
Address	0xbe896a6ebd3d3ea465c8c34168483ee94043136a
Network	BSC
Symbol	ODWIF
Decimals	9
Total Supply	100,000,000,000,000,000
Badge Eligibility	Yes

Audit Updates

Initial Audit	26 May 2024

Source Files

Filename	SHA256
ODWIF.sol	d0af6aed208646fb77f53fac713e5dc7cf0cb3613901bdffaa0ff74035f3399f

Findings Breakdown

Critical	0
Medium	0
Minor / Informative	11

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	11	0	0	0

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	ODWIF.sol#L333,334
Status	Unresolved

Description

Division of decimal (fixed point) numbers can result in rounding errors due to the way that division is implemented in Solidity. Thus, it may produce issues with precise calculations with decimal numbers.

Solidity represents decimal numbers as integers, with the decimal point implied by the number of decimal places specified in the type (e.g. decimal with 18 decimal places). When a division is performed with decimal numbers, the result is also represented as an integer, with the decimal point implied by the number of decimal places in the type. This can lead to rounding errors, as the result may not be able to be accurately represented as an integer with the specified number of decimal places.

Hence, the splitted shares will not have the exact precision and some funds may not be calculated as expected.

```
if(buyAllocation > 0 || sellAllocation > 0)
internalSwap((contractTokenBalance * (buyAllocation +
sellAllocation)) / 100);
if(liquidityAllocation > 0)
{swapAndLiquify(contractTokenBalance * liquidityAllocation /
100);}
```

Recommendation

The team is advised to take into consideration the rounding results that are produced from the solidity calculations. The contract could calculate the subtraction of the divided funds in the last calculation in order to avoid the division rounding issue.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	ODWIF.sol#L241
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
lpPair
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	ODWIF.sol#L279
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
function setNoFeeWallet(address account, bool enabled) public
onlyOwner {
    _noFee[account] = enabled;
}
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

PAV - Pair Address Validation

Criticality	Minor / Informative
Location	ODWIF.sol#L308
Status	Unresolved

Description

The contract is missing address validation in the pair address argument. The absence of validation reveals a potential vulnerability, as it lacks proper checks to ensure the integrity and validity of the pair address provided as an argument. The pair address is a parameter used in certain methods of decentralized exchanges for functions like token swaps and liquidity provisions.

The absence of address validation in the pair address argument can introduce security risks and potential attacks. Without proper validation, if the owner's address is compromised, the contract may lead to unexpected behavior like loss of funds.

```
function changeLpPair(address newPair) external onlyOwner {
    isLpPair[newPair] = true;
    emit _changePair(newPair);
}
```

Recommendation

To mitigate the risks associated with the absence of address validation in the pair address argument, it is recommended to implement comprehensive address validation mechanisms. A recommended approach could be to verify pair existence in the decentralized application. Prior to interacting with the pair address contract, perform checks to verify the existence and validity of the contract at the provided address. This can be achieved by querying the provider's contract or utilizing external libraries that provide contract verification services.

RED - Redudant Event Declaration

Criticality	Minor / Informative
Location	ODWIF.sol#L213,215
Status	Unresolved

Description

The contract uses events that are not emitted within the contract's functions. As a result, these declared events are redundant and serve no purpose within the contract's current implementation.

```
event _changeThreshold(uint256 newThreshold);
event _changeFees(uint256 buy, uint256 sell);
```

Recommendation

To optimize contract performance and efficiency, it is advisable to regularly review and refactor the codebase, removing the unused event declarations. This proactive approach not only streamlines the contract, reducing deployment and execution costs but also enhances readability and maintainability.

RCS - Redundant Conditional Statement

Criticality	Minor / Informative
Location	ODWIF.sol#L155,156
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The variable `_totalSupply` is initialized on the contract's constructor, which means its value is always greater than zero. Hence, the conditional statement is redundant.

```
function totalSupply() external pure override returns (uint256) { if (_totalSupply == 0) { revert(); } return _totalSupply; }
function decimals() external pure override returns (uint8) { if (_totalSupply == 0) { revert(); } return _decimals; }
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

RIC - Redundant If Condition

Criticality	Minor / Informative
Location	ODWIF.sol#L333,334,357
Status	Unresolved

Description

The `_transfer` function includes a check to ensure that the amount being transferred is greater than zero. Within the `takeTaxes` function, there is an additional check `if (feeAmount > 0)` to determine whether the `feeAmount` is greater than zero. This additional check in the `takeTaxes` function is redundant. The `_transfer` function already ensures that the amount is greater than zero. Additionally, if the fee is zero, the `takeTaxes` function will return the amount before reaching the if statement. As a result, the condition `if (feeAmount > 0)` will always evaluate to `true` when the function proceeds past the initial return statement, making this check unnecessary and potentially confusing.

```
function takeTaxes(address from, bool isbuy, bool issell,
uint256 amount) internal returns (uint256) {
    uint256 fee;
    if (isbuy)  fee = buyfee;  else if (issell)  fee = sellfee;
else  fee = transferfee;
    if (fee == 0)  return amount;
    uint256 feeAmount = amount * fee / fee_denominator;
    if (feeAmount > 0) {

        balance[address(this)] += feeAmount;
        emit Transfer(from, address(this), feeAmount);

    }
    return amount - feeAmount;
}
```

Furthermore, the conditions `if(buyAllocation > 0 || sellAllocation > 0)` and `if(liquidityAllocation > 0)` in the `_transfer` function are also redundant. The allocation values for `buyAllocation`, `sellAllocation`, and `liquidityAllocation` are set as constants (40, 40, and 20 respectively) and cannot change. These checks are therefore unnecessary, as the conditions will always be true given the fixed allocation values.

```
uint256 private buyAllocation = 40;
uint256 private sellAllocation = 40;
uint256 private liquidityAllocation = 20;

if(buyAllocation > 0 || sellAllocation > 0)
internalSwap((contractTokenBalance * (buyAllocation +
sellAllocation)) / 100);
if(liquidityAllocation > 0)
{swapAndLiquify(contractTokenBalance * liquidityAllocation /
100);}
```

Recommendation

It is recommended to remove the redundant `if (feeAmount > 0)` statement from the `takeTaxes` function. This check does not affect the actual implementation and adds unnecessary complexity to the code. Additionally, the conditions `if(buyAllocation > 0 || sellAllocation > 0)` and `if(liquidityAllocation > 0)` should be removed from the `_transfer` function. Simplifying the function by removing this redundant condition will make the contract easier to read, understand, and maintain.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	ODWIF.sol#L279,348
Status	Unresolved

Description

The contract modifies the state of the following variables without checking if their current value is the same as the one given as an argument. As a result, the contract performs redundant storage writes, when the provided parameter matches the current state of the variables, leading to unnecessary gas consumption and inefficiencies in contract execution.

```
function setNoFeeWallet(address account, bool enabled) public
onlyOwner {
    _noFee[account] = enabled;
}

function changeWallets(address newBuy, address newDev) external
onlyOwner {
    require(newBuy != address(0),"Freddy: Address Zero");
    require(newDev != address(0),"Freddy: Address Zero");
    marketingAddress = payable(newBuy);
    devWallet = payable(newDev);
    emit _changeWallets(newBuy, newDev);
}
```

Recommendation

The team is advised to implement additional checks within to prevent redundant storage writes when the provided argument matches the current state of the variables. By incorporating statements to compare the new values with the existing values before proceeding with any state modification, the contract can avoid unnecessary storage operations, thereby optimizing gas usage.

RSD - Redundant Swap Duplication

Criticality	Minor / Informative
Location	ODWIF.sol#L371,406
Status	Unresolved

Description

The contract contains multiple swap methods that individually perform token swaps and transfer promotional amounts to specific addresses and features. This redundant duplication of code introduces unnecessary complexity and increases dramatically the gas consumption. By consolidating these operations into a single swap method, the contract can achieve better code readability, reduce gas costs, and improve overall efficiency.

```
function swapAndLiquify(uint256 contractTokenBalance) internal
inSwapFlag {
        uint256 firstmath = contractTokenBalance / 2;
        uint256 secondMath = contractTokenBalance - firstmath;

        uint256 initialBalance = address(this).balance;

        address[] memory path = new address[](2);
        path[0] = address(this);
        path[1] = swapRouter.WETH();

        if (_allowances[address(this)][address(swapRouter)] !=
type(uint256).max) {
            _allowances[address(this)][address(swapRouter)] =
type(uint256).max;
        }

        try
swapRouter.swapExactTokensForETHSupportingFeeOnTransferTokens(
            firstmath,
            0,
            path,
            address(this),
            block.timestamp) {} catch {return;}

        uint256 newBalance = address(this).balance -
initialBalance;

        try swapRouter.addLiquidityETH{value: newBalance}(
            address(this),
            secondMath,
            0,
            0,
            DEAD,
            block.timestamp
        ){} catch {return;}

        emit SwapAndLiquify();
    }

    function internalSwap(uint256 contractTokenBalance)
internal inSwapFlag {

        address[] memory path = new address[](2);
        path[0] = address(this);
        path[1] = swapRouter.WETH();

        if (_allowances[address(this)][address(swapRouter)] !=
type(uint256).max) {
            _allowances[address(this)][address(swapRouter)] =
```

```
type(uint256).max;
        }

        try
swapRouter.swapExactTokensForETHSupportingFeeOnTransferTokens(
            contractTokenBalance,
            0,
            path,
            address(this),
            block.timestamp
        ) {} catch {
            return;
        }
        bool success;

        uint256 mktAmount = address(this).balance * 25 / 40;
        uint256 devAmount = address(this).balance * 15 / 40;

        if(mktAmount > 0) (success,) =
marketingAddress.call{value: mktAmount, gas: 35000}("");
        if(devAmount > 0) (success,) = devWallet.call{value:
devAmount, gas: 35000}("");
    }
```

Recommendation

A more optimized approach could be adopted to perform the token swap operation once for the total amount of tokens and distribute the proportional amounts to the corresponding addresses, eliminating the need for separate swaps.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	ODWIF.sol#L187,188,189
Status	Unresolved

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
uint256 private buyAllocation = 40
uint256 private sellAllocation = 40
uint256 private liquidityAllocation = 20
```

Recommendation

Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	ODWIF.sol#L76,193,194,196,209,210,211,212,213,214,215,292,297
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
function WETH() external pure returns (address);
string constant private _name = "OnedollarwifHat"
string constant private _symbol = "ODWIF"
uint8 constant private _decimals = 9
event _enableTrading();
event _setPresaleAddress(address account, bool enabled);
event _toggleCanSwapFees(bool enabled);
event _changePair(address newLpPair);
event _changeThreshold(uint256 newThreshold);
event _changeWallets(address newBuy, address newSell);
event _changeFees(uint256 buy, uint256 sell);

...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.
Find more information on the Solidity documentation
https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

Functions Analysis

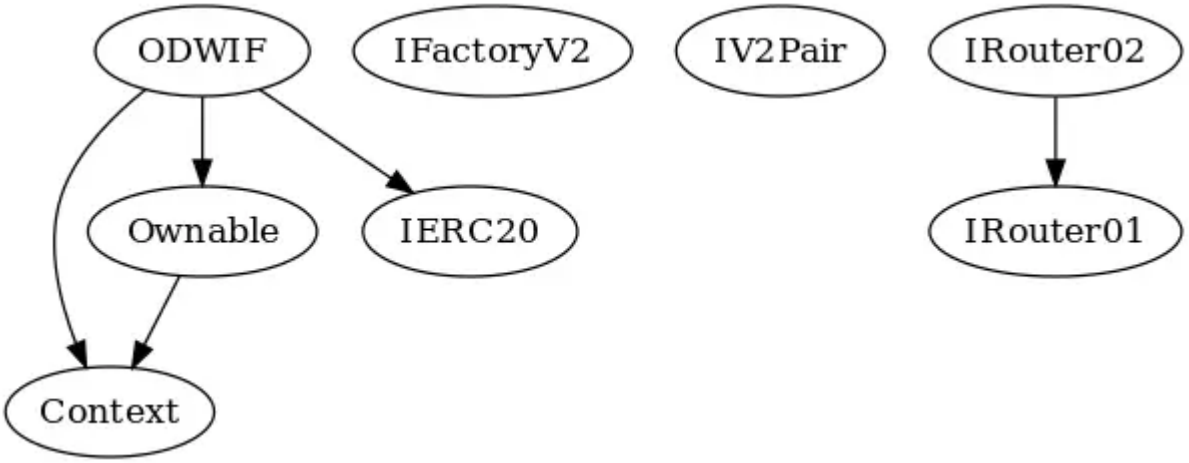
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
		Public	✓	-
	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_setOwner	Private	✓	
IFactoryV2	Interface			
	getPair	External		-
	createPair	External	✓	-
IV2Pair	Interface			
	factory	External		-
	getReserves	External		-

	sync	External	✓	-
IRouter01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidityETH	External	Payable	-
	addLiquidity	External	✓	-
	swapExactETHForTokens	External	Payable	-
	getAmountsOut	External		-
	getAmountsIn	External		-
IRouter02	Interface	IRouter01		
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokens	External	✓	-
IERC20	Interface			
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-

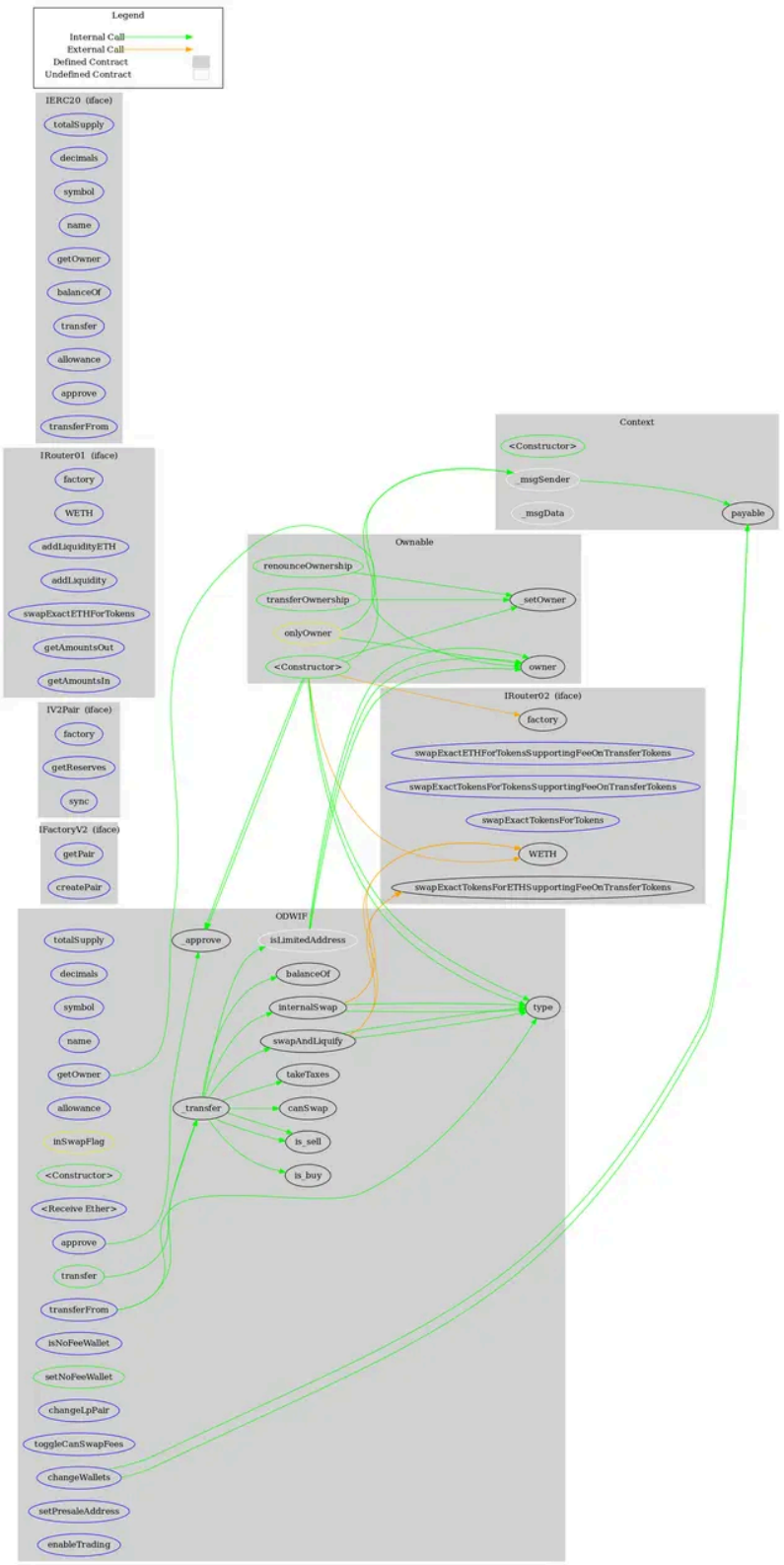
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
ODWIF	Implementation	Context, Ownable, IERC20		
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	allowance	External		-
	balanceOf	Public		-
		Public	✓	-
		External	Payable	-
	transfer	Public	✓	-
	approve	External	✓	-
	_approve	Internal	✓	
	transferFrom	External	✓	-
	isNoFeeWallet	External		-
	setNoFeeWallet	Public	✓	onlyOwner
	isLimitedAddress	Internal		

	is_buy	Internal		
	is_sell	Internal		
	canSwap	Internal		
	changeLpPair	External	✓	onlyOwner
	toggleCanSwapFees	External	✓	onlyOwner
	_transfer	Internal	✓	
	changeWallets	External	✓	onlyOwner
	takeTaxes	Internal	✓	
	swapAndLiquify	Internal	✓	inSwapFlag
	internalSwap	Internal	✓	inSwapFlag
	setPresaleAddress	External	✓	onlyOwner
	enableTrading	External	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IERC20 (iface)
totalSupply
decimals
symbol
name
getOwner
balanceOf
transfer
allowance
approve
transferFrom
IRouter01 (iface)
factory
WETH
addLiquidityETH
addLiquidity
swapExactETHForTokens
getAmountsOut
getAmountsIn
IV2Pair (iface)
factory
getReserves
sync
IFactoryV2 (iface)
getPair
createPair
Context
<Constructor>
_msgSender
_msgData
payable
Ownable
renounceOwnership
transferOwnership
onlyOwner
<Constructor>
setOwner
owner
IRouter02 (iface)
factory
swapExactETHForTokensSupportingFeeOnTransferTokens
swapExactTokensForTokensSupportingFeeOnTransferTokens
swapExactTokensForTokens
WETH
swapExactTokensForETHSupportingFeeOnTransferTokens
ODWIF
totalSupply
decimals
symbol
name
getOwner
allowance
inSwapFlag
<Constructor>
<Receive Ether>
approve
transfer
transferFrom
isNoFeeWallet
setNoFeeWallet
changeLpPair
toggleCanSwapFees
changeWallets
setPresaleAddress
enableTrading
_approve
_transfer
isLimitedAddress
balanceOf
internalSwap
swapAndLiquify
takeTaxes
canSwap
is_sell
is_buy
type

Summary

OnedollarwifHat contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. OnedollarwifHat is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues. The contract ownership has been renounced.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io