Cyberscope

Audit Report
OLEFY

March 2023

Network	BSC
Address	0x969839175ABD1280d041C14A478d6FBaCa637218
Audited by	© cyberscope

Table of Contents

Review

Contract Name	Olefy
Compiler Version	v0.8.15+commit.e14f2714
Optimization	200 runs
Explorer	https://bscscan.com/address/0x969839175abd1280d041c14a478d6fbaca637218
Address	0x969839175abd1280d041c14a478d6fbaca637218
Network	BSC
Symbol	Olefy
Decimals	18
Total Supply	100,000,000

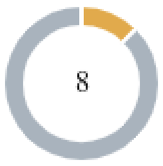
Audit Updates

Initial Audit	31 Mar 2023

Source Files

Filename	SHA256
Olefy.sol	5112dfe30db9e713d1c28257eaecc8da7eecab6f62a1dc4805ec3377c62a3c8c

Findings Breakdown



●	Critical	0
●	Medium	1
●	Minor / Informative	7

	Severity	Unresolved	Acknowledged	Resolved	Other
●	Critical	0	0	0	0
●	Medium	1	0	0	0
●	Minor / Informative	7	0	0	0

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	OCTD	Transfers Contract's Tokens	Unresolved
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	ULTW	Transfers Liquidity to Team Wallet	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

OCTD - Transfers Contract's Tokens

Criticality	Minor / Informative
Location	Olefy.sol#L386
Status	Unresolved

Description

The contract owner has the authority to claim all the balance of the contract. The owner may take advantage of it by calling the `manualSend` function.

```
function manualSend() external onlyOwner {
    payable(marketingFeeReceiver).transfer(address(this).balance);
    _basicTransfer(
        address(this),
        marketingFeeReceiver,
        balanceOf[address(this)]
    );
}
```

Recommendation

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

Diagnostics

🔴 Critical 🟠 Medium ⚪ Minor / Informative

Severity	Code	Description	Status
🟠	ZD	Zero Division	Unresolved
⚪	DDP	Decimal Division Precision	Unresolved
⚪	L02	State Variables could be Declared Constant	Unresolved
⚪	L04	Conformance to Solidity Naming Conventions	Unresolved
⚪	L07	Missing Events Arithmetic	Unresolved
⚪	L12	Using Variables before Declaration	Unresolved
⚪	L16	Validate Variable Setters	Unresolved

ZD - Zero Division

Criticality	Medium
Location	Olefy.sol#L323
Status	Unresolved

Description

The contract is using variables that may be set to zero as denominators. This can lead to unpredictable and potentially harmful results, such as a transaction revert.

The `totalETHFee` variable is equal to `totalSellFee`, which is the sum of the following variables:

- `sellLiquidityFee`
- `sellExpansaoFee`
- `sellPionnerFee`
- `sellMarketingFee`

All these fees can be set to zero. As a result, the transaction will revert.

```
uint256 amountToLiquify = (swapThreshold * sellLiquidityFee) /
    (totalETHFee * 2);
```

Recommendation

It is important to handle division by zero appropriately in the code to avoid unintended behavior and to ensure the reliability and safety of the contract. The contract should ensure that the divisor is always non-zero before performing a division operation. It should prevent the variables to be set to zero, or should not allow the execution of the corresponding statements.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	Olefy.sol#L343,346,349,351
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.

```
uint256 amountBNBLiquidity = (amountBNB * sellLiquidityFee) /
    (totalETHFee * 2);

uint256 amountBNBMarketing = (amountBNB * sellMarketingFee) /
    totalETHFee;

uint256 amountBNBExpansao = (amountBNB * sellExpansaoFee) / totalETHFee;

uint256 amountBNBPionner = (amountBNB * sellPionnerFee) / totalETHFee;
```

Recommendation

The contract could calculate the subtraction of the divided funds in the last calculation in order to avoid the division rounding issue.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	Olefy.sol#L101,114
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 transferMultiplier = 25;
bool public burnEnabled = true;
```

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	Olefy.sol#L37,82,90,91,100,104,108,109,111,395,399,414,415,416,417,418,419,420,421,450,462,469,476,510,511,512,513,514
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);
mapping(address => mapping(address => uint256)) _allowances
uint256 public ExpansaoFee = 20
uint256 public PionnerFee = 20;
uint256 public constant feeDenominator = 1000;
uint256 LiquidifyGas = 500000;
address public ExpansaoReceiver;
address public PionnerReceiver;
address public WBNB = router.WETH();
address _pair,

...
```

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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	Olefy.sol#L482,487
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
txbnbGas = gas;
LiquidifyGas = gas;
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L12 - Using Variables before Declaration

Criticality	Minor / Informative
Location	Olefy.sol#L354
Status	Unresolved

Description

The contract is using a variable before the declaration. This is usually happening either if it has not been declared yet or if the variable has been declared in a different scope. It is not a good practice to use a local variable before it has been declared.

```
bool success,
```

Recommendation

By declaring local variables before using them, contract ensures that it operates correctly. It's important to be aware of this rule when working with local variables, as using a variable before it has been declared can lead to unexpected behavior and can be difficult to debug.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	Olefy.sol#L466,473,477
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
marketingFeeReceiver = _marketingFeeReceiver;
ExpansaoReceiver = _autoExpansaoReceiver;
PionnerReceiver = _PionnersReceiver;
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

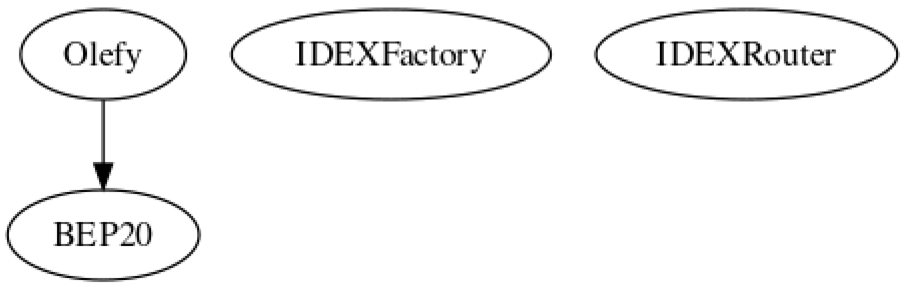
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
BEP20	Interface			
	balanceOf	External		-
	transfer	External	✓	-
	approve	External	✓	-
	transferFrom	External	✓	-
IDEXFactory	Interface			
	createPair	External	✓	-
IDEXRouter	Interface			
	factory	External		-
	WETH	External		-
	addLiquidityETH	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
Olefy	Implementation	BEP20		
		Public	✓	-

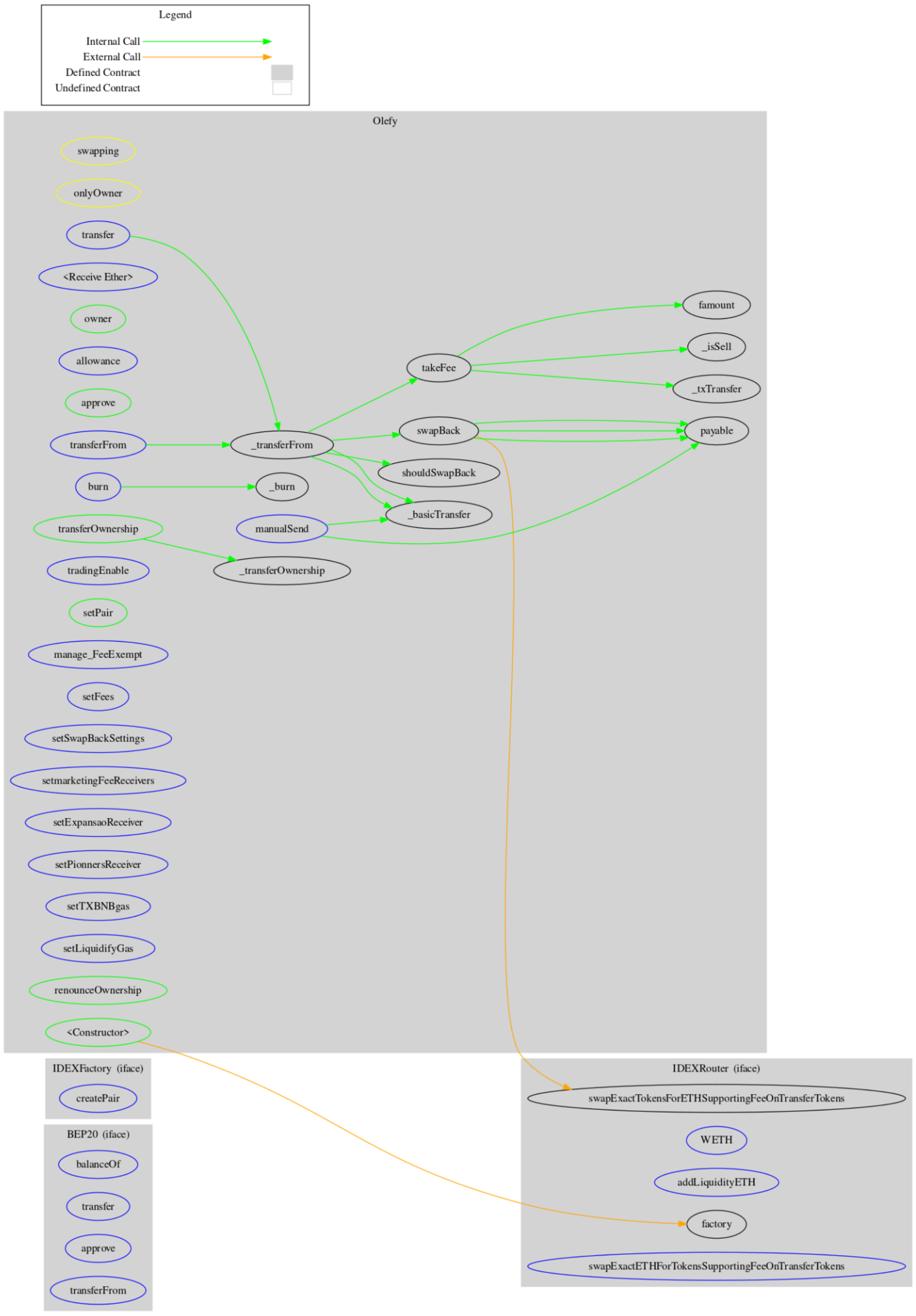
		External	Payable	-
	owner	Public		-
	allowance	External		-
	approve	Public	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	_transferFrom	Internal	✓	
	_basicTransfer	Internal	✓	
	_isSell	Internal		
	burn	External	✓	-
	_burn	Internal	✓	
	famount	Internal		
	takeFee	Internal	✓	
	_txTransfer	Internal	✓	
	shouldSwapBack	Internal		
	tradingEnable	External	✓	onlyOwner
	swapBack	Internal	✓	swapping
	manualSend	External	✓	onlyOwner
	setPair	Public	✓	onlyOwner
	manage_FeeExempt	External	✓	onlyOwner
	setFees	External	✓	onlyOwner
	setSwapBackSettings	External	✓	onlyOwner
	setmarketingFeeReceivers	External	✓	onlyOwner

	setExpansaoReceiver	External	✓	onlyOwner
	setPionnersReceiver	External	✓	onlyOwner
	setTXBNBgas	External	✓	onlyOwner
	setLiquidifyGas	External	✓	onlyOwner
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Olefy
swapping
onlyOwner
transfer
<Receive Ether>
owner
allowance
approve
transferFrom
burn
transferOwnership
tradingEnable
setPair
manage_FeeExempt
setFees
setSwapBackSettings
setmarketingFeeReceivers
setExpansaoReceiver
setPionnersReceiver
setTXBNBgas
setLiquidifyGas
renounceOwnership
<Constructor>
_transferFrom
_burn
manualSend
_transferOwnership
takeFee
swapBack
shouldSwapBack
_basicTransfer
famount
_isSell
_txTransfer
payable
IDEXFactory (iface)
createPair
BEP20 (iface)
balanceOf
transfer
approve
transferFrom
IDEXRouter (iface)
swapExactTokensForETHSupportingFeeOnTransferTokens
WETH
addLiquidityETH
factory
swapExactETHForTokensSupportingFeeOnTransferTokens

Summary

OLEFY contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. There are some functions that can be abused by the owner like draining the contract's tokens. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats. There is also a limit of max 12% buy/sell fees, and 3% transfer fees.

Disclaimer

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