Cyberscope

Audit Report

Odesis

May 2023

Network BSC

Address 0xa68D77D85b24257adae0bC9Ca96edcBCba5F03Ea

Audited by © cyberscope

Table of Contents

Table of Contents **1**
Review **4**
Audit Updates 4
Source Files 4
Findings Breakdown **5**
Analysis **6**
ST - Stops Transactions 7
Description 7
Recommendation 7
Diagnostics **8**
TSD - Total Supply Diversion 10
Description 10
Recommendation 10
RCS - Redundant Code Segment 11
Description 11
Recommendation 11
ZD - Zero Division 12
Description 12
Recommendation 12
LED - Lacking Event Details 13
Description 13
Recommendation 13
ISA - Inaccurate Swap Amount 14
Description 14
Recommendation 14
AOI - Arithmetic Operations Inconsistency 15
Description 15
Recommendation 15
TUU - Time Units Usage 16
Description 16
Recommendation 16
RED - Redundant Event Declaration 17
Description 17
Recommendation 17
PVC - Price Volatility Concern 18
Description 18
Recommendation 18
DDP - Decimal Division Precision 19
Description 19

Recommendation 20
RVD - Redundant Variable Declaration 21
Description 21
Recommendation 21
MAL - Misused Algorithmic Logic 22
Description 22
Recommendation 22
MC - Missing Check 23
Description 23
Recommendation 23
RS - Redundant Swaps 24
Description 24
Recommendation 24
RSML - Redundant SafeMath Library 25
Description 25
Recommendation 25
RSK - Redundant Storage Keyword 26
Description 26
Recommendation 26
L02 - State Variables could be Declared Constant 27
Description 27
Recommendation 27
L04 - Conformance to Solidity Naming Conventions 28
Description 28
Recommendation 29
L05 - Unused State Variable 30
Description 30
Recommendation 30
L07 - Missing Events Arithmetic 31
Description 31
Recommendation 31
L09 - Dead Code Elimination 32
Description 32
Recommendation 32
L12 - Using Variables before Declaration 33
Description 33
Recommendation 33
L13 - Divide before Multiply Operation 34
Description 34
Recommendation 34
L14 - Uninitialized Variables in Local Scope 35
Description 35

Recommendation 35
L15 - Local Scope Variable Shadowing 36
Description 36
Recommendation 36
L16 - Validate Variable Setters 37
Description 37
Recommendation 37
L19 - Stable Compiler Version 38
Description 38
Recommendation 38
L20 - Succeeded Transfer Check 39
Description 39
Recommendation 39
Functions Analysis **40**
Inheritance Graph **52**
Flow Graph **53**
Summary **54**
Disclaimer **55**
About Cyberscope **56**

Review

Contract Name	Odesis
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	200 runs
Explorer	https://bscscan.com/address/0xa68d77d85b24257adae0bc9ca96edcbcba5f03ea
Address	0xa68d77d85b24257adae0bc9ca96edcbcba5f03ea
Network	BSC
Symbol	$ODS
Decimals	18
Total Supply	1,000,000,000

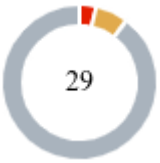
Audit Updates

Initial Audit	16 May 2023

Source Files

Filename	SHA256
Odesis.sol	aa6af84f9cb777c540ee47239d631bb8b89f50f1fd2a52e44df2de75f68ffd78

Findings Breakdown



Critical	1
Medium	2
Minor / Informative	26

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	2	0	0	0
Minor / Informative	26	0	0	0

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
	ST	Stops Transactions	Unresolved
	OCTD	Transfers Contract's Tokens	Passed
	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

ST - Stops Transactions

Criticality	Minor / Informative
Location	Odesis.sol#L1406,1427
Status	Unresolved

Description

Initially, the contract does not allow the non-excluded addresses to transfer tokens. The restriction can be resumed once the contract owner enables them.

```
if (!canTransferBeforeTradingIsEnabled[from]) {
    require(tradingEnabled, "Trading has not yet been enabled");
}
```

The contract owner has the authority to stop the sales and transfers for all users excluding the addresses that are excluded from fees. The contract owner may take advantage of it by setting the `stakingEnabled` to true.

```
require(
    stakingUntilDate[from] <= block.timestamp,
    "Tokens are staked and locked!"
);
```

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
	TSD	Total Supply Diversion	Unresolved
	RCS	Redundant Code Segment	Unresolved
	ZD	Zero Division	Unresolved
	LED	Lacking Event Details	Unresolved
	ISA	Inaccurate Swap Amount	Unresolved
	AOI	Arithmetic Operations Inconsistency	Unresolved
	TUU	Time Units Usage	Unresolved
	RED	Redundant Event Declaration	Unresolved
	PVC	Price Volatility Concern	Unresolved
	DDP	Decimal Division Precision	Unresolved
	RVD	Redundant Variable Declaration	Unresolved
	MAL	Misused Algorithmic Logic	Unresolved
	MC	Missing Check	Unresolved
	RS	Redundant Swaps	Unresolved

●	RSML	Redundant SafeMath Library	Unresolved
●	RSK	Redundant Storage Keyword	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L12	Using Variables before Declaration	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

TSD - Total Supply Diversion

Criticality	Critical
Location	Odesis.sol#L1557
Status	Unresolved

Description

The total supply of a token is the total number of tokens that have been created, while the balances of individual accounts represent the number of tokens that an account owns. The total supply and the balances of individual accounts are two separate concepts that are managed by different variables in a smart contract. These two entities should be equal to each other.

In the contract, the amount that is added to the total supply does not equal the amount that is added to the balances. As a result, the sum of balances is diverse from the total supply.

```
if (to == DEAD) {
    _totalSupply = _totalSupply.sub(amount);
}
```

Recommendation

The total supply and the balance variables are separate and independent from each other. The total supply represents the total number of tokens that have been created, while the balance mapping stores the number of tokens that each account owns. The sum of balances should always equal the total supply.

RCS - Redundant Code Segment

Criticality	Medium
Location	Odesis.sol#L1653
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 contract sends the amount of dividends that are bought and sold to the dividend tracker contract. The variables `dividendsFromBuy` and `dividendsFromSell` are initialized with no given value. Hence, both variables will have their default value which is zero. Since their value will be zero, the `if` statement will never execute. As a result, this code segment is redundant.

```
uint256 dividends;
uint256 dividendsFromBuy;
uint256 dividendsFromSell;

dividends = dividendsFromBuy.add(dividendsFromSell);

if (dividends > 0) {
    (successOp, ) = address(dividendTracker).call{value: dividends}("");
    success = success && successOp;
}
```

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.

ZD - Zero Division

Criticality	Medium
Location	Odesis.sol#L1495
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.

If the variable `swapTokensAtAmount` is set to zero, then the variables `buyAmount` and `sellAmount` will may not have accumulated fees before the contract enters the swapping stage and their value will be zero. As a result, the contract will execute a division by zero causing the transaction to revert.

```
uint256 totalBuySell = buyAmount.add(sellAmount);
uint256 swapAmountBought = contractTokenBalance
    .mul(buyAmount)
    .div(totalBuySell);
```

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.

LED - Lacking Event Details

Criticality	Minor / Informative
Location	Odesis.sol#L1700
Status	Unresolved

Description

The contract emits the following event during the swap functionality. However, the information provided in the event is insufficient to identify the cause of a failed transfer. Furthermore, the `dividends` variable consistently holds the value of zero, as explained in detail in the RCS section. Therefore, these aspects could be improved for better clarity and usefulness.

```
emit SendDividends(
    dividends,
    marketingPayout + teamPayout + botPayout + restFundPayout,
    success
);
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the event will provide improved details for better clarity and usefulness.

ISA - Inaccurate Swap Amount

Criticality	Minor / Informative
Location	Odesis.sol#L1507
Status	Unresolved

Description

The contract calculates the amount of tokens that have been bought and sold in preparation for the swap functionality. The contract is using the `liquidityFee` variable in both expressions, while the `swapSellTokens` should be calculated using the `sellLiquidityFee`. As a result, the contract will swap an inaccurate amount of tokens.

```
uint256 swapBuyTokens = swapAmountBought
    .mul(liquidityFee)
    .div(totalBuyFees);

uint256 swapSellTokens = swapAmountSold
    .mul(liquidityFee)
    .div(totalSellFees);
```

Recommendation

The team is advised to use the `sellLiquidityFee` for the calculation of the sold tokens.

AOI - Arithmetic Operations Inconsistency

Criticality	Minor / Informative
Location	Odesis.sol#L1481
Status	Unresolved

Description

The contract demonstrates an inconsistency in arithmetic operations. To ensure consistency and safety, all operations should be executed using one source of truth. In the given code segment, the addition operation (`marketingFees + botFees + teamFees + restFundsFees`) is performed using a native arithmetic expression, which is not aligned with the rest of the contract that utilizes the SafeMath library methods.

```
uint256 totalFees = liquidityFee.add(marketingFees + botFees + teamFees +
restFundsFees);
```

Recommendation

The team is advised to use only one source of truth for the arithmetic operations.

TUU - Time Units Usage

Criticality	Minor / Informative
Location	Odesis.sol#L1153
Status	Unresolved

Description

The contract is using arbitrary numbers to form time-related values. As a result, it decreases the readability of the codebase and prevents the compiler to optimize the source code.

```
require(value <= 300, "cooldown timer cannot exceed 5 minutes");
```

Recommendation

It is a good practice to use the time units reserved keywords like `seconds` , `minutes` , `hours` , `days` , `weeks` and `years` to process time-related calculations.

It's important to note that these time units are simply a shorthand notation for representing time in seconds, and do not have any effect on the actual passage of time or the execution of the contract. The time units are simply a convenience for expressing time in a more human-readable form.

RED - Redundant Event Declaration

Criticality	Minor / Informative
Location	Odesis.sol#L952
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 contract declares the event `UpdateDividendTracker` but it is not being used in the contract. As a result, it is redundant.

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

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.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	Odesis.sol#L1190
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `swapTokensAtAmount` sets a threshold where the contract will trigger the swap functionality. If the variable is set to a big number, then the contract will swap a huge amount of tokens for ETH.

It is important to note that the price of the token representing it, can be highly volatile. This means that the value of a price volatility swap involving Ether could fluctuate significantly at the triggered point, potentially leading to significant price volatility for the parties involved.

```
function setSwapTriggerAmount(uint256 amount) public onlyOwner {
    swapTokensAtAmount = amount * (10 ** 18);
}
```

Recommendation

The contract could ensure that it will not sell more than a reasonable amount of tokens in a single transaction. A suggested implementation could check that the maximum amount should be less than a fixed percentage of the total supply. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	Odesis.sol#L1434,1496
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 swapAmountBought = contractTokenBalance
    .mul(buyAmount)
    .div(totalBuySell);
uint256 swapAmountSold = contractTokenBalance
    .mul(sellAmount)
    .div(totalBuySell);

uint256 swapBuyTokens = swapAmountBought
    .mul(liquidityFee)
    .div(totalBuyFees);

uint256 swapSellTokens = swapAmountSold
    .mul(liquidityFee)
    .div(totalSellFees);

...
uint256 fromBuy = tokens.mul(buyAmount).div(totalAmount);
uint256 fromSell = tokens.mul(sellAmount).div(totalAmount);
```

Recommendation

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

RVD - Redundant Variable Declaration

Criticality	Minor / Informative
Location	Odesis.sol#L1524
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 contract declares the `burntokens` variable and subtracts it from the amount. It is initialized with a value of zero, hence it does not impact the result of the subtraction in any way. As a result, the `burntokens` variable is redundant.

```
uint256 burntokens;

amount = amount.sub(fees + burntokens);
```

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.

MAL - Misused Algorithmic Logic

Criticality	Minor / Informative
Location	Odesis.sol#L1528
Status	Unresolved

Description

The algorithmic flow does not follow the required business logic.

The contract checks if the transaction is a sale and adds the fees to the `sellAmount` variable, else it adds the fees to the `buyAmount` variable. The else block contains transactions that are not buys, such as simple transfers. As a result, the `buyAmount` will have a misleading value.

```
if (isSelling) {
    sellAmount = sellAmount.add(fees);
} else {
    buyAmount = buyAmount.add(fees);
}
```

Recommendation

The code segment should be reshaped to match the business logic.

MC - Missing Check

Criticality	Minor / Informative
Location	Odesis.sol#L1631
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues.

The tokens should be less than or equal to the balance of the contract.

```
function forceSwapAndSendDividends(uint256 tokens) public onlyOwner {
    tokens = tokens * (10 ** 18);
    uint256 totalAmount = buyAmount.add(sellAmount);
    uint256 fromBuy = tokens.mul(buyAmount).div(totalAmount);
    uint256 fromSell = tokens.mul(sellAmount).div(totalAmount);

    swapAndSendDividends(tokens);

    buyAmount = buyAmount.sub(fromBuy);
    sellAmount = sellAmount.sub(fromSell);
}
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

RS - Redundant Swaps

Criticality	Minor / Informative
Location	Odesis.sol#L1190
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 `swapTokensAtAmount` variable can be set to zero since there is no input validation. As a result, the contract will perform redundant swaps.

```
function setSwapTriggerAmount(uint256 amount) public onlyOwner {
    swapTokensAtAmount = amount * (10 ** 18);
}
```

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. The contract could embody a check for not allowing setting the `swapTokensAtAmount` less than a reasonable amount. A suggested implementation could check that the minimum amount should be more than a fixed percentage of the total supply.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	Odesis.sol
Status	Unresolved

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert to underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, and overhead and increases gas consumption unnecessarily.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change at https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

RSK - Redundant Storage Keyword

Criticality	Minor / Informative
Location	Odesis.sol#L2158,2163,2173,2179
Status	Unresolved

Description

The contract uses the `storage` keyword in a view function. The `storage` keyword is used to persist data on the contract's storage. View functions are functions that do not modify the state of the contract and do not perform any actions that cost gas (such as sending a transaction). As a result, the use of the `storage` keyword in view functions is redundant.

```
Map storage map
```

Recommendation

It is generally considered good practice to avoid using the `storage` keyword in view functions because it is unnecessary and can make the code less readable.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	Odesis.sol#L886
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.

```
address public DEAD = 0x000000000000000000000000000000000000dEaD
```

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	Odesis.sol#L49,51,82,199,541,743,807,812,818,824,886,899,1146,1729,1867
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 DOMAIN_SEPARATOR() external view returns (bytes32);
function PERMIT_TYPEHASH() external pure returns (bytes32);
function MINIMUM_LIQUIDITY() external pure returns (uint256);
uint256 internal _totalSupply
function WETH() external pure returns (address);
uint256 internal constant magnitude = 2 ** 128
address _owner
address public DEAD = 0x000000000000000000000000000000000000dEaD
address payable public MarketingWallet
uint256 GWEI
Odesis public OdesisContract
address _account
```

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.

L05 - Unused State Variable

Criticality	Minor / Informative
Location	Odesis.sol#L488
Status	Unresolved

Description

An unused state variable is a state variable that is declared in the contract, but is never used in any of the contract's functions. This can happen if the state variable was originally intended to be used, but was later removed or never used.

Unused state variables can create clutter in the contract and make it more difficult to understand and maintain. They can also increase the size of the contract and the cost of deploying and interacting with it.

```
int256 private constant MAX_INT256 = ~(int256(1) << 255)
```

Recommendation

To avoid creating unused state variables, it's important to carefully consider the state variables that are needed for the contract's functionality, and to remove any that are no longer needed. This can help improve the clarity and efficiency of the contract.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	Odesis.sol#L1148,1154,1164,1191,1639
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.

```
gasPriceLimit = GWEI * 1 gwei
cooldowntimer = value
maxWallet = value
swapTokensAtAmount = amount * (10 ** 18)
buyAmount = buyAmount.sub(fromBuy)
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	Odesis.sol#L519,780,834,2158
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function abs(int256 a) internal pure returns (int256) {
        require(a != MIN_INT256);
        return a < 0 ? -a : a;
    }

...
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L12 - Using Variables before Declaration

Criticality	Minor / Informative
Location	Odesis.sol#L1539,1540,1541
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.

```
uint256 iterations
uint256 claims
uint256 lastProcessedIndex
```

Recommendation

By declaring local variables before using them, the 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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	Odesis.sol#L1496,1499,1503,1507,1670,1672,1674,1675,1676
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause a loss of prediction.

```
feePortions = address(this).balance.div(_completeFees)
uint256 botPayout = buyBotFee.add(sellBotFee) * feePortions
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	Odesis.sol#L1400,1401,1402,1403,1404,1524,1539,1540,1541,1654,1655,1668,2047,2107
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
uint256 marketingFees
uint256 teamFees
uint256 restFundsFees
uint256 liquidityFee
uint256 botFees
uint256 burntokens
uint256 iterations
uint256 claims
uint256 lastProcessedIndex
uint256 dividendsFromBuy
uint256 dividendsFromSell
uint256 feePortions
bool success
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	Odesis.sol#L753,754,1756
Status	Unresolved

Description

Local scope variable shadowing occurs when a local variable with the same name as a variable in an outer scope is declared within a function or code block. When this happens, the local variable "shadows" the outer variable, meaning that it takes precedence over the outer variable within the scope in which it is declared.

```
string memory _name
string memory _symbol
```

Recommendation

It's important to be aware of shadowing when working with local variables, as it can lead to confusion and unintended consequences if not used correctly. It's generally a good idea to choose unique names for local variables to avoid shadowing outer variables and causing confusion.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	Odesis.sol#L458,1761,2035
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.

```
_owner = msgSender
defaultToken = token
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	Odesis.sol#L3
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.19;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	Odesis.sol#L2076
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
OdesisContract.transfer(account, received)
```

Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
IUniswapV2Pair	Interface			
	name	External		-
	symbol	External		-
	decimals	External		-
	totalSupply	External		-
	balanceOf	External		-
	allowance	External		-
	approve	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	DOMAIN_SEPARATOR	External		-
	PERMIT_TYPEHASH	External		-
	nonces	External		-
	permit	External	✓	-

	MINIMUM_LIQUIDITY	External		-
	factory	External		-
	token0	External		-
	token1	External		-
	getReserves	External		-
	price0CumulativeLast	External		-
	price1CumulativeLast	External		-
	kLast	External		-
	mint	External	✓	-
	burn	External	✓	-
	swap	External	✓	-
	skim	External	✓	-
	sync	External	✓	-
	initialize	External	✓	-
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-

	setFeeToSetter	External	✓	-
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-

	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
DividendPayingTokenOptionalInterface	Interface			
	withdrawableDividendOf	External		-
	withdrawnDividendOf	External		-
	accumulativeDividendOf	External		-
DividendPayingTokenInterface	Interface			
	dividendOf	External		-
	distributeDividends	External	Payable	-
	withdrawDividend	External	✓	-
SafeMath	Library			

	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
	mod	Internal		
	mod	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
SafeMathInt	Library			
	mul	Internal		
	div	Internal		
	sub	Internal		
	add	Internal		
	abs	Internal		
	toUint256Safe	Internal		

SafeMathUint	Library			
	toInt256Safe	Internal		
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-
	swapTokensForExactETH	External	✓	-
	swapExactTokensForETH	External	✓	-
	swapETHForExactTokens	External	Payable	-
	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-
	getAmountsOut	External		-
	getAmountsIn	External		-

IUniswapV2Router02	Interface	IUniswapV2Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
DividendPayingToken	Implementation	ERC20, DividendPayingTokenInterface, DividendPayingTokenOptionalInterface		
		Public	✓	ERC20
		External	Payable	-
	distributeDividends	Public	Payable	-
	withdrawDividend	Public	✓	-
	_withdrawDividendOfUser	Internal	✓	
	dividendOf	Public		-
	withdrawableDividendOf	Public		-
	withdrawnDividendOf	Public		-
	accumulativeDividendOf	Public		-
	_transfer	Internal	✓	
	_mint	Internal	✓	

	_burn	Internal	✓	
	_setBalance	Internal	✓	
Odesis	Implementation	ERC20, Ownable		
		Public	✓	ERC20
	decimals	Public		-
		External	Payable	-
	updateStakingAmounts	Public	✓	onlyOwner
	enableTrading	External	✓	onlyOwner
	setPresaleWallet	External	✓	onlyOwner
	setExcludeFees	Public	✓	onlyOwner
	setExcludeDividends	Public	✓	onlyOwner
	setIncludeDividends	Public	✓	onlyOwner
	setCanTransferBefore	External	✓	onlyOwner
	setLimitsInEffect	External	✓	onlyOwner
	setGasPriceLimit	External	✓	onlyOwner
	setcooldowntimer	External	✓	onlyOwner
	setmaxWallet	External	✓	onlyOwner
	enableStaking	Public	✓	onlyOwner
	stake	Public	✓	-
	setSwapTriggerAmount	Public	✓	onlyOwner
	enableSwapAndLiquify	Public	✓	onlyOwner
	setAutomatedMarketMakerPair	Public	✓	onlyOwner

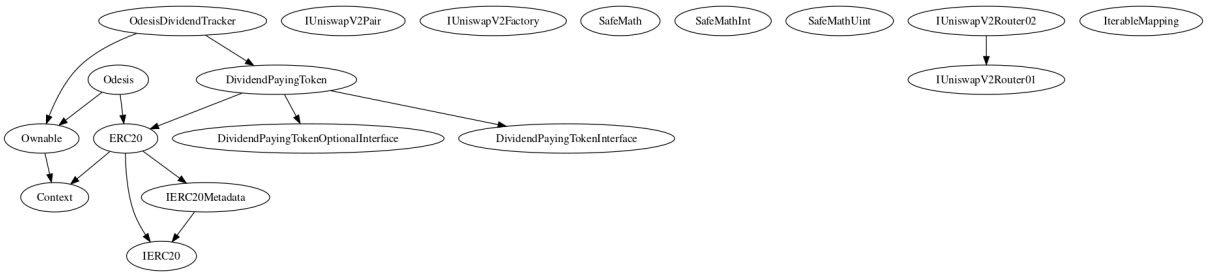
	setAllowCustomTokens	Public	✓	onlyOwner
	setAllowAutoReinvest	Public	✓	onlyOwner
	_setAutomatedMarketMakerPair	Private	✓	
	updateGasForProcessing	Public	✓	onlyOwner
	transferAdmin	Public	✓	onlyOwner
	updateTransferFee	Public	✓	onlyOwner
	updateFees	Public	✓	onlyOwner
	getStakingInfo	External		-
	getTotalDividendsDistributed	External		-
	isExcludedFromFees	Public		-
	withdrawableDividendOf	Public		-
	dividendTokenBalanceOf	Public		-
	getAccountDividendsInfo	External		-
	getAccountDividendsInfoAtIndex	External		-
	processDividendTracker	External	✓	-
	claim	External	✓	-
	getLastProcessedIndex	External		-
	getNumberOfDividendTokenHolders	External		-
	setAutoClaim	External	✓	-
	setReinvest	External	✓	-
	setDividendsPaused	External	✓	onlyOwner
	isExcludedFromAutoClaim	External		-
	isReinvest	External		-

	_transfer	Internal	✓	
	getStakingBalance	Private		
	swapAndLiquify	Private	✓	
	swapTokensForEth	Private	✓	
	updatePayoutToken	Public	✓	onlyOwner
	getPayoutToken	Public		-
	setMinimumTokenBalanceForAutoDividends	Public	✓	onlyOwner
	setMinimumTokenBalanceForDividends	Public	✓	onlyOwner
	addLiquidity	Private	✓	
	forceSwapAndSendDividends	Public	✓	onlyOwner
	swapAndSendDividends	Private	✓	
OdesisDividendTracker	Implementation	DividendPayingToken, Ownable		
		Public	✓	DividendPayingToken
	decimals	Public		-
	name	Public		-
	symbol	Public		-
	_transfer	Internal		
	withdrawDividend	Public		-
	isExcludedFromAutoClaim	External		onlyOwner
	isReinvest	External		onlyOwner
	setAllowCustomTokens	External	✓	onlyOwner

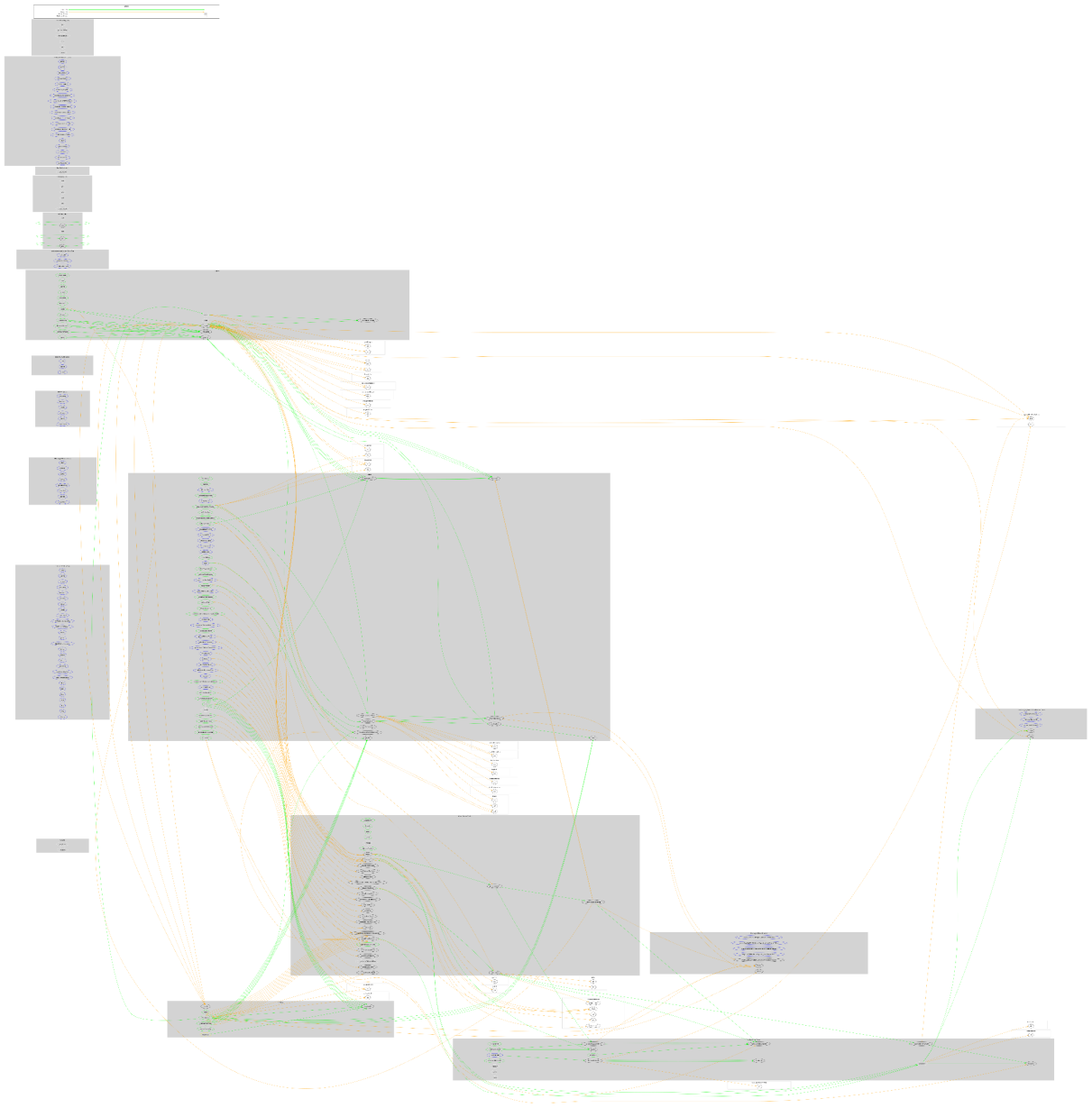
	setAllowAutoReinvest	External	✓	onlyOwner
	excludeFromDividends	External	✓	onlyOwner
	includeFromDividends	External	✓	onlyOwner
	setAutoClaim	External	✓	onlyOwner
	setReinvest	External	✓	onlyOwner
	setMinimumTokenBalanceForAutoDividends	External	✓	onlyOwner
	setMinimumTokenBalanceForDividends	External	✓	onlyOwner
	setDividendsPaused	External	✓	onlyOwner
	getLastProcessedIndex	External		-
	getNumberOfTokenHolders	External		-
	getAccount	Public		-
	getAccountAtIndex	Public		-
	setBalance	External	✓	onlyOwner
	process	Public	✓	-
	processAccount	Public	✓	onlyOwner
	updateUniswapV2Router	Public	✓	onlyOwner
	updatePayoutToken	Public	✓	onlyOwner
	getPayoutToken	Public		-
	_reinvestDividendOfUser	Private	✓	
	_withdrawDividendOfUser	Internal	✓	
IterableMapping	Library			
	get	Internal		

	getIndexOfKey	Internal		
	getKeyAtIndex	Internal		
	size	Internal		
	set	Internal	✓	
	remove	Internal	✓	

Inheritance Graph



Flow Graph



Summary

Odesis 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 stopping transactions. 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 15% sell, 5% buy, and 2% transfer fees.

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