Cyberscope

Audit Report

# Zetta Word

November 2024

| | |
|---|---|
| Network | BSC |
| Address | 0x8AaCC38933007eC530c552007E210B4667749DF1 |
| 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 |
|---|---|---|---|
| ● | CR | Code Repetition | Unresolved |
| ● | DDP | Decimal Division Precision | Unresolved |
| ● | MVN | Misleading Variable Naming | Unresolved |
| ● | PLPI | Potential Liquidity Provision Inadequacy | Unresolved |
| ● | PNR | Privileges Not Revoked | Unresolved |
| ● | RAO | Redundant Addition Operation | Unresolved |
| ● | RCL | Redundant Calculation Logic | Unresolved |
| ● | RCS | Redundant Conditional Statement | Unresolved |
| ● | RRA | Redundant Repeated Approvals | Unresolved |
| ● | RSD | Redundant Swap Duplication | Unresolved |
| ● | UOD | Unnecessary Override Declaration | Unresolved |
| ● | L04 | Conformance to Solidity Naming Conventions | Unresolved |
| ● | L13 | Divide before Multiply Operation | Unresolved |
| ● | L17 | Usage of Solidity Assembly | Unresolved |

# Table of Contents

## Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

| Severity | Likelihood / Impact of Exploitation |
|---|---|
| ● Critical | Highly Likely / High Impact |
| ● Medium | Less Likely / High Impact or Highly Likely/ Lower Impact |
| ● Minor / Informative | Unlikely / Low to no Impact |

## Review

| | |
|---|---|
| **Contract Name** | zetta_word |
| **Compiler Version** | v0.8.25+commit.b61c2a91 |
| **Optimization** | 200 runs |
| **Explorer** | https://bscscan.com/address/0x8aacc38933007ec530c552007e210b4667749df1 |
| **Address** | 0x8aacc38933007ec530c552007e210b4667749df1 |
| **Network** | BSC |
| **Symbol** | Z |
| **Decimals** | 18 |
| **Total Supply** | 1,000,000,000 |
| **Badge Eligibility** | Yes |

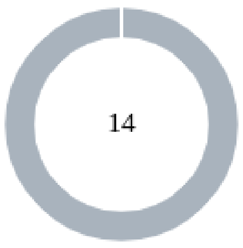
## Audit Updates

| | |
|---|---|
| **Initial Audit** | 26 Nov 2024 |

## Source Files

| **Filename** | SHA256 |
|---|---|
| **Token.sol** | 936341012fb5aac0bda8d95c9622130aaaf0ef2d1c85fc6bcc68bb0b760c9c97 |

## Findings Breakdown



| | | |
|---|---|---|
| ● | Critical | 0 |
| ● | Medium | 0 |
| ● | Minor / Informative | 14 |

| Severity | | Unresolved | Acknowledged | Resolved | Other |
|---|---|---|---|---|---|
| ● | Critical | 0 | 0 | 0 | 0 |
| ● | Medium | 0 | 0 | 0 | 0 |
| ● | Minor / Informative | 14 | 0 | 0 | 0 |

## CR - Code Repetition

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | Token.sol#L169,214 |
| **Status** | Unresolved |

### Description

The contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

```
   function taxbuyFeesSetup(uint16 _buyFee, uint16 _sellFee, uint16
_transferFee) public onlyOwner {
        totalFees[0] = totalFees[0] - taxbuyFees[0] + _buyFee;
        totalFees[1] = totalFees[1] - taxbuyFees[1] + _sellFee;
        totalFees[2] = totalFees[2] - taxbuyFees[2] + _transferFee;
        if (totalFees[0] > 2500 || totalFees[1] > 2500 || totalFees[2] >
2500) revert CannotExceedMaxTotalFee(totalFees[0], totalFees[1],
totalFees[2]);

        taxbuyFees = [_buyFee, _sellFee, _transferFee];

        emit WalletTaxFeesUpdated(1, _buyFee, _sellFee, _transferFee);
    }

    function liquidityFeesSetup(uint16 _buyFee, uint16 _sellFee, uint16
_transferFee) public onlyOwner {
        totalFees[0] = totalFees[0] - liquidityFees[0] + _buyFee;
        totalFees[1] = totalFees[1] - liquidityFees[1] + _sellFee;
        totalFees[2] = totalFees[2] - liquidityFees[2] + _transferFee;
        if (totalFees[0] > 2500 || totalFees[1] > 2500 || totalFees[2] >
2500) revert CannotExceedMaxTotalFee(totalFees[0], totalFees[1],
totalFees[2]);

        liquidityFees = [_buyFee, _sellFee, _transferFee];

        emit LiquidityFeesUpdated(_buyFee, _sellFee, _transferFee);
    }
```

## Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help reduce the complexity and size of the contract. For instance, the contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places.

## DDP - Decimal Division Precision

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | Token.sol#L280 |
| **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.

```
fees = amount * totalFees[txType] / 10000;
amount -= fees;

_taxbuyPending += fees * taxbuyFees[txType] / totalFees[txType];

_liquidityPending += fees * liquidityFees[txType] / totalFees[txType];
```

### 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.

## MVN - Misleading Variable Naming

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | Token.sol#L305 |
| **Status** | Unresolved |

### Description

The contract is performing a token swap for ETH using the `_swapTokensForCoin` function and then assigns the current contract ETH balance to the `coinsReceived` variable. However, this naming is misleading because the variable does not represent the actual amount of coins received from the swap but rather the total ETH balance of the contract after the operation. The actual coins received should be calculated as the difference between the current balance and the previous balance before the swap.

```
_swapTokensForCoin(token2Swap);
uint256 coinsReceived = address(this).balance;
```

### Recommendation

It is recommended to rename the variable to something more indicative, such as `currentBalance`, or alternatively, compute the actual coins received by subtracting the previous ETH balance from the new ETH balance after the swap. This will improve the clarity and accuracy of the code, reducing potential confusion for developers and auditors.

## PLPI - Potential Liquidity Provision Inadequacy

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | Token.sol#L134 |
| **Status** | Unresolved |

### Description

The contract operates under the assumption that liquidity is consistently provided to the pair between the contract's token and the native currency. However, there is a possibility that liquidity is provided to a different pair. This inadequacy in liquidity provision in the main pair could expose the contract to risks. Specifically, during eligible transactions, where the contract attempts to swap tokens with the main pair, a failure may occur if liquidity has been added to a pair other than the primary one. Consequently, transactions triggering the swap functionality will result in a revert.

```
    function _swapTokensForCoin(uint256 tokenAmount) private {
        address[] memory path = new address[](2);
        path[0] = address(this);
        path[1] = routerV2.WETH();

        _approve(address(this), address(routerV2), tokenAmount);


routerV2.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,
0, path, address(this), block.timestamp);
    }
```

### Recommendation

The team is advised to implement a runtime mechanism to check if the pair has adequate liquidity provisions. This feature allows the contract to omit token swaps if the pair does not have adequate liquidity provisions, significantly minimizing the risk of potential failures.

Furthermore, the team could ensure the contract has the capability to switch its active pair in case liquidity is added to another pair.

Additionally, the contract could be designed to tolerate potential reverts from the swap functionality, especially when it is a part of the main transfer flow. This can be achieved by executing the contract's token swaps in a non-reversible manner, thereby ensuring a more resilient and predictable operation.

## PNR - Privileges Not Revoked

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | Token.sol#L98,160 |
| **Status** | Unresolved |

### Description

The contract grants specific addresses certain privileges, providing operational flexibility. However, the current implementation does not automatically remove these privileges when an address's status changes. Consequently, the privileges remain active for addresses that no longer hold the specified roles or ownership. This oversight can lead to unintended discrepancies in privileges and poses potential security risks.

```
    constructor()
        ERC20(unicode"zetta word", unicode"Z")
        Ownable(msg.sender)
    {
        ...
        excludeFromFees(supplyRecipient, true);
        ...
        }

    function taxbuyAddressSetup(address _newAddress) public onlyOwner {
        if (_newAddress == address(0)) revert
InvalidTaxRecipientAddress(address(0));

        taxbuyAddress = _newAddress;
        excludeFromFees(_newAddress, true);

        emit WalletTaxAddressUpdated(1, _newAddress);
    }
```

## Recommendation

It is advised to modify the contract to include functionality that revokes privileges from old addresses and grants them to new ones whenever there is a change in roles or ownership. This method will ensure consistent and equitable distribution of privileges while preserving the integrity and security of the contract.

## RAO - Redundant Addition Operation

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | Token.sol#L157,301 |
| **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 includes code segments with redundant addition operations with zero. These additions are unnecessary, as adding any number to zero yields the same result. This redundancy does not contribute to the logic and may create confusion or code maintenance challenges.

```
function getAllPending() public view returns (uint256) {
    return 0 + _taxbuyPending + _liquidityPending;
}

uint256 token2Swap = 0 + _taxbuyPending;
```

### 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.

## RCL - Redundant Calculation Logic

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | Token.sol#L301 |
| **Status** | Unresolved |

### Description

The contract is performing redundant calculations by setting `tokens2Swap` equal to `_taxBuyPending` and then using both variables in subsequent calculations, such as multiplying `coinsReceived` by `_taxBuyPending` and dividing it by `tokens2Swap`. Since `tokens2Swap` is already equal to `_taxBuyPending`, this operation is unnecessarily repetitive and adds no functional value, potentially leading to confusion and inefficiencies.

```
uint256 token2Swap = 0 + _taxbuyPending;
bool success = false;

...
uint256 coinsReceived = address(this).balance;

uint256 taxbuyPortion = coinsReceived * _taxbuyPending / token2Swap;
```

### Recommendation

It is recommended to simplify the logic by removing redundant variables or calculations. Instead of setting `tokens2Swap` to `_taxBuyPending` and then reusing both, directly use `_taxBuyPending` in the calculations where applicable. This will enhance the code's clarity, reduce complexity, and improve gas efficiency.

## RCS - Redundant Conditional Statement

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | Token.sol#L250,300 |
| **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 conditional statement `if (false || _taxbuyPending > 0)` in the contract is redundant. The `_taxbuyPending > 0` condition has already been checked earlier within the transfer function, making the code segment unnecessary and serving no purpose.

```
if (false || _taxbuyPending > 0) {
    ...
}
```

Additionally, the `_setAMMPair` function contains a control flow block that executes no code. As a result, the code segment is redundant.

```
if (isPair) {
}
```

### 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.

## RRA - Redundant Repeated Approvals

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | Token.sol#L139,201 |
| **Status** | Unresolved |

### Description

The contract is designed to `approve` token transfers during the contract's operation by calling the _approve function before specific operations. This approach results in additional gas costs since the approval process is repeated for every operation execution, leading to inefficiencies and increased transaction expenses.

```
...
_approve(address(this), address(routerV2), tokenAmount);
...
_approve(address(this), address(routerV2), tokenAmount);
```

### Recommendation

Since the approved address is a trusted third-party source, it is recommended to optimize the contract by approving the maximum amount of tokens once in the initial set of the variable, rather than before each operation. This change will reduce the overall gas consumption and improve the efficiency of the contract.

## RSD - Redundant Swap Duplication

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | Token.sol#L304,319 |
| **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.

```
   _swapTokensForCoin(token2Swap);
    ...
}
if (_liquidityPending > 0) {
    _swapAndLiquify(_liquidityPending);
    _liquidityPending = 0;
}
```

### 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.

## UOD - Unnecessary Override Declaration

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | Token.sol#L112 |
| **Status** | Unresolved |

### Description

The contract is currently implementing an override of the decimals function, which simply returns the value 18. This override is redundant since the extending token contract already specifies 18 decimals as its standard. In the context of ERC-20 tokens, 18 decimals is a common default, and overriding this function to return the same value adds unnecessary complexity to the contract. This redundancy does not contribute to the functionality of the contract and could potentially lead to confusion about the necessity of this override.

```
    function decimals() public pure override returns (uint8) {
        return 18;
    }
```

### Recommendation

It is recommended to remove the `override` keyword from the decimals function, assuming the extending token contract already defines 18 decimals. This action will simplify the contract by eliminating redundant code, thereby enhancing its clarity and efficiency. The removal of this unnecessary override will not impact the contract's functionality but will contribute to a cleaner and more maintainable codebase.

## L04 - Conformance to Solidity Naming Conventions

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | Token.sol#L32,53,108,144,160,169,214,241 |
| **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.

```
contract zetta_word is ERC20, ERC20Burnable, Ownable2Step, Initializable
{

    using SafeERC20Remastered for IERC20;

    uint16 public swapThresholdRatio;

...

    function _afterTokenUpdate(address from, address to, uint256 amount)
        internal
    {
    }
}


...
```

## 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/stable/style-guide.html#naming-conventions.

## L13 - Divide before Multiply Operation

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | Token.sol#L280,283,285 |
| **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 loss of prediction.

```
fees = amount * totalFees[txType] / 10000;

_taxbuyPending += fees * taxbuyFees[txType] / totalFees[txType];
```

### 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.

## L17 - Usage of Solidity Assembly

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | Token.sol#L88 |
| **Status** | Unresolved |

### Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly { if iszero(extcodesize(caller())) { revert(0, 0) } }
```

### Recommendation

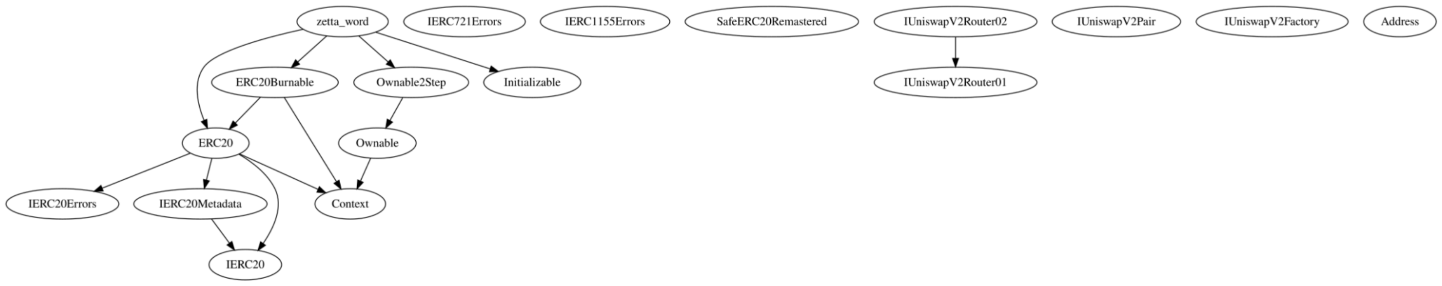
It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

## Functions Analysis

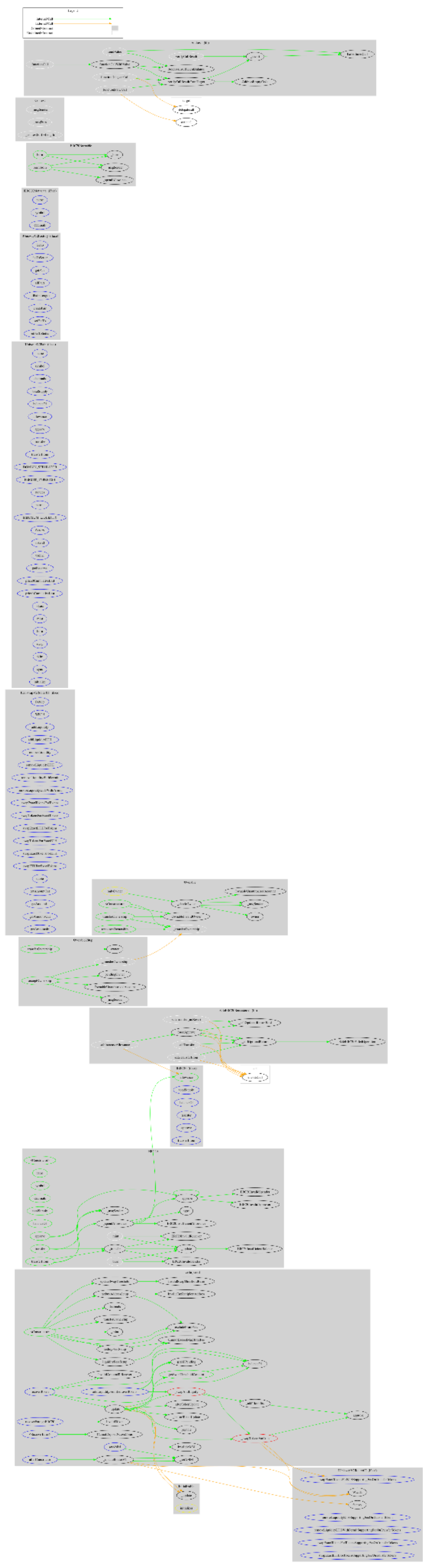
| Contract | Type | Bases | | |
|---|---|---|---|---|
| | **Function Name** | **Visibility** | **Mutability** | **Modifiers** |
| | | | | |
| **zetta_word** | Implementation | ERC20, ERC20Burnable, Ownable2Step, Initializable | | |
| | | Public | ✓ | ERC20 Ownable |
| | afterConstructor | External | ✓ | initializer |
| | decimals | Public | | - |
| | recoverToken | External | ✓ | onlyOwner |
| | recoverForeignERC20 | External | ✓ | onlyOwner |
| | | External | Payable | - |
| | _swapTokensForCoin | Private | ✓ | |
| | updateSwapThreshold | Public | ✓ | onlyOwner |
| | getSwapThresholdAmount | Public | | - |
| | getAllPending | Public | | - |
| | taxbuyAddressSetup | Public | ✓ | onlyOwner |
| | taxbuyFeesSetup | Public | ✓ | onlyOwner |
| | _swapAndLiquify | Private | ✓ | |
| | _addLiquidity | Private | ✓ | |
| | addLiquidityFromLeftoverTokens | External | ✓ | - |
| | liquidityFeesSetup | Public | ✓ | onlyOwner |

| | | | | |
|---|---|---|---|---|
| | excludeFromFees | Public | ✓ | onlyOwner |
| | _updateRouterV2 | Private | ✓ | |
| | setAMM | External | ✓ | onlyOwner |
| | _setAMM | Private | ✓ | |
| | _update | Internal | ✓ | |
| | _beforeTokenUpdate | Internal | | |
| | _afterTokenUpdate | Internal | ✓ | |

## Inheritance Graph



## Flow Graph



## Summary

Zetta Word contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. Zetta Word is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues. The contract Owner can access some admin functions that can not be used in a malicious way to disturb the users’ transactions. There is also a limit of max 25% 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**

cyberscope.io