CertiFi™

CRYPTO AGENCY

@certifiagency

Audit Report

February 28, 2024

Bot Bros Sniper Bot Audit Report

www.certifiagency.com

Tableof Contents.

1. Vulnerability Classification and Severity

Description

To enhance navigability, the document is organized in descending order of severity for easy reference. Issues are categorized as ✓ *Fixed*, ⚠ *Pending Fix*, or ☒ *Won't Fix*, indicating their current status. ☒ *Won't Fix* denotes that the team is aware of the issue but has chosen not to resolve it. Issues labeled as ⚠ *Pending Fix* state that the bug is yet to be resolved. Additionally, each issue's severity is assessed based on the risk of exploitation or the potential for other unexpected or unsafe behavior.

● Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

● High

High-severity vulnerabilities pose a significant risk to both the Smart Contract and the organization. They can lead to user fund losses, may have conditional requirements, and are challenging to exploit.

● Medium

The issue affects the ability of the contract to operate in a way that doesn't significantly hinder its behavior.

● Low

The issue has minimal impact on the contract's ability to operate.

● Gas

This category deals with optimizing code and refactoring to conserve gas.

● Informational

The issue does not affect the contract's operational capability but is considered good practice to address.

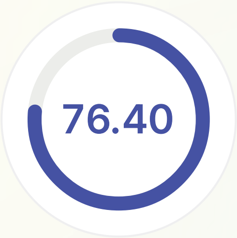
2. ExecutiveSummary

SniperBot- BuyContract

Uploaded Solidity File(s)

Published on 29 Feb 2024

Language	Audit Methodology	Website
Solidity	**Static Scanning**	-
Publishers/Owner Name	Organization	Contact Email
-	-	-



Security Score is AVERAGE

The CertiFi score is calculated based on lines of code and weights assigned to each issue depending on the severity and confidence. To improve your score, view the detailed result and leverage the remediation solutions provided.

This report has been prepared for Sniper Bot - Buy Contract to scan and discover vulnerabilities and safe coding practices in their smart contract including the libraries used by the contract that are not officially recognized. CertiFi runs a comprehensive static analysis on the Solidity code and finds vulnerabilities ranging from minor gas optimizations to major vulnerabilities leading to the loss of funds. The coverage scope pays attention to all the informational and critical vulnerabilities with over (100) modules. The scanning and auditing process covers the following areas:

Various common and uncommon attack vectors will be investigated to ensure that the smart contracts are secure from malicious actors. The scanner modules find and flag issues related to Gas optimizations that help in reducing the overall Gas cost It scans and evaluates the codebase against industry best practices and standards to ensure compliance It makes sure that the officially recognized libraries used in the code are secure and up to date

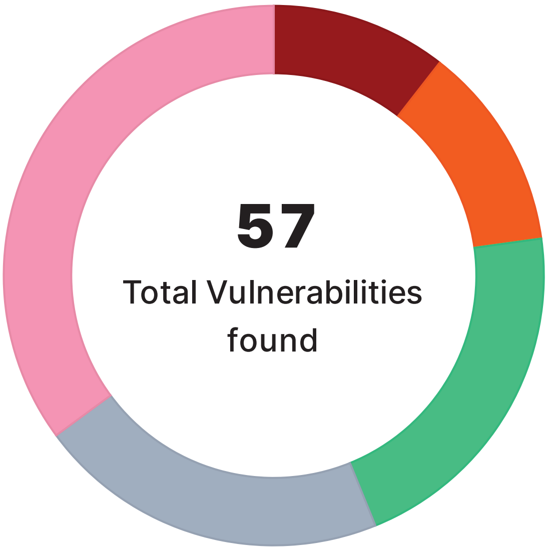
The CertiFi Team recommends running regular audit scans to identify any vulnerabilities that are introduced after Sniper Bot - Buy Contract introduces new features or refactors the code.

3. Findings Summary

Sniper Bot - Buy Contract

File Scan

Security Score	Scan duration	Lines of code
76.40/100	1 secs	483



6	7	0	12	12	20
Crit	High	Med	Low	Info	Gas

ACTIONTAKEN

0	0	0	57
✓ *Fixed*	*False Positive*	*Won't Fix*	⚠ *Pending Fix*

Bug ID	Severity	Bug Type	Detection Method	Line No	Status
SSP_4274_15	● Critical	INCORRECT ACCESS CONTROL	Automated	L78 - L117	⚠ *Pending Fix*
SSP_4274_16	● Critical	INCORRECT ACCESS CONTROL	Automated	L127 - L178	⚠ *Pending Fix*
SSP_4274_17	● Critical	INCORRECT ACCESS CONTROL	Automated	L186 - L231	⚠ *Pending Fix*
SSP_4274_18	● Critical	INCORRECT ACCESS CONTROL	Automated	L240 - L280	⚠ *Pending Fix*
SSP_4274_19	● Critical	INCORRECT ACCESS CONTROL	Automated	L290 - L342	⚠ *Pending Fix*
SSP_4274_20	● Critical	INCORRECT ACCESS CONTROL	Automated	L350 - L396	⚠ *Pending Fix*
SSP_4274_30	● High	REENTRANCY	Automated	L78 - L117	⚠ *Pending Fix*
SSP_4274_31	● High	REENTRANCY	Automated	L127 - L178	⚠ *Pending Fix*
SSP_4274_32	● High	REENTRANCY	Automated	L186 - L231	⚠ *Pending Fix*
SSP_4274_33	● High	REENTRANCY	Automated	L240 - L280	⚠ *Pending Fix*
SSP_4274_34	● High	REENTRANCY	Automated	L290 - L342	⚠ *Pending Fix*
SSP_4274_35	● High	REENTRANCY	Automated	L350 - L396	⚠ *Pending Fix*
SSP_4274_9	● High	UNCHECKED TRANSFER	Automated	L143 - L143	⚠ *Pending Fix*
SSP_4274_9	● High	UNCHECKED TRANSFER	Automated	L196 - L196	⚠ *Pending Fix*
SSP_4274_9	● High	UNCHECKED TRANSFER	Automated	L307 - L307	⚠ *Pending Fix*

Bug ID	Severity	Bug Type	Detection Method	Line No	Status
SSP_4274_9	● High	UNCHECKED TRANSFER	Automated	L361 - L361	⚠ *Pending Fix*
SSP_4274_43	● Low	EVENT BASED REENTRANCY	Automated	L78 - L117	⚠ *Pending Fix*
SSP_4274_44	● Low	EVENT BASED REENTRANCY	Automated	L127 - L178	⚠ *Pending Fix*
SSP_4274_45	● Low	EVENT BASED REENTRANCY	Automated	L186 - L231	⚠ *Pending Fix*
SSP_4274_46	● Low	EVENT BASED REENTRANCY	Automated	L240 - L280	⚠ *Pending Fix*
SSP_4274_47	● Low	EVENT BASED REENTRANCY	Automated	L290 - L342	⚠ *Pending Fix*
SSP_4274_48	● Low	EVENT BASED REENTRANCY	Automated	L350 - L396	⚠ *Pending Fix*
SSP_4274_23	● Low	USE OF FLOATING PRAGMA	Automated	L2 - L2	⚠ *Pending Fix*
SSP_4274_12	● Low	MISSING EVENTS	Automated	L454 - L458	⚠ *Pending Fix*
SSP_4274_13	● Low	MISSING EVENTS	Automated	L460 - L464	⚠ *Pending Fix*
SSP_4274_14	● Low	MISSING EVENTS	Automated	L470 - L481	⚠ *Pending Fix*
SSP_4274_25	● Low	OUTDATED COMPILER VERSION	Automated	L2 - L2	⚠ *Pending Fix*
SSP_4274_5	● Low	USE OWNABLE2STEP	Automated	L10 - L10	⚠ *Pending Fix*
SSP_4274_54	● Informational	BLOCK VALUES AS A PROXY FOR TIME	Automated	L107 - L107	⚠ *Pending Fix*
SSP_4274_55	● Informational	BLOCK VALUES AS A PROXY FOR TIME	Automated	L152 - L152	⚠ *Pending Fix*
SSP_4274_55	● Informational	BLOCK VALUES AS A PROXY FOR TIME	Automated	L205 - L205	⚠ *Pending Fix*
SSP_4274_56	● Informational	BLOCK VALUES AS A PROXY FOR TIME	Automated	L267 - L267	⚠ *Pending Fix*
SSP_4274_57	● Informational	BLOCK VALUES AS A PROXY FOR TIME	Automated	L316 - L316	⚠ *Pending Fix*

Bug ID	Severity	Bug Type	Detection Method	Line No	Status
SSP_4274_57	● Informational	BLOCK VALUES AS A PROXY FOR TIME	Automated	L370 - L370	⚠ *Pending Fix*
SSP_4274_2	● Informational	MISSING PAYABLE IN CALL FUNCTION	Automated	L162 - L162	⚠ *Pending Fix*
SSP_4274_3	● Informational	MISSING PAYABLE IN CALL FUNCTION	Automated	L165 - L165	⚠ *Pending Fix*
SSP_4274_4	● Informational	MISSING PAYABLE IN CALL FUNCTION	Automated	L168 - L168	⚠ *Pending Fix*
SSP_4274_2	● Informational	MISSING PAYABLE IN CALL FUNCTION	Automated	L215 - L215	⚠ *Pending Fix*
SSP_4274_3	● Informational	MISSING PAYABLE IN CALL FUNCTION	Automated	L218 - L218	⚠ *Pending Fix*
SSP_4274_4	● Informational	MISSING PAYABLE IN CALL FUNCTION	Automated	L221 - L221	⚠ *Pending Fix*
SSP_4274_2	● Informational	MISSING PAYABLE IN CALL FUNCTION	Automated	L326 - L326	⚠ *Pending Fix*
SSP_4274_3	● Informational	MISSING PAYABLE IN CALL FUNCTION	Automated	L329 - L329	⚠ *Pending Fix*
SSP_4274_4	● Informational	MISSING PAYABLE IN CALL FUNCTION	Automated	L332 - L332	⚠ *Pending Fix*
SSP_4274_2	● Informational	MISSING PAYABLE IN CALL FUNCTION	Automated	L380 - L380	⚠ *Pending Fix*
SSP_4274_3	● Informational	MISSING PAYABLE IN CALL FUNCTION	Automated	L383 - L383	⚠ *Pending Fix*
SSP_4274_4	● Informational	MISSING PAYABLE IN CALL FUNCTION	Automated	L386 - L386	⚠ *Pending Fix*
SSP_4274_6	● Informational	MISSING UNDERSCORE IN NAMING VARIABLES	Automated	L18 - L18	⚠ *Pending Fix*
SSP_4274_7	● Informational	MISSING UNDERSCORE IN NAMING VARIABLES	Automated	L20 - L20	⚠ *Pending Fix*
SSP_4274_8	● Informational	MISSING UNDERSCORE IN NAMING VARIABLES	Automated	L407 - L423	⚠ *Pending Fix*
SSP_4274_36	● Informational	UNUSED RECEIVE FALLBACK	Automated	L466 - L468	⚠ *Pending Fix*
SSP_4274_27	● Informational	USE CALL INSTEAD OF TRANSFER OR SEND	Automated	L480 - L480	⚠ *Pending Fix*

Bug ID	Severity	Bug Type	Detection Method	Line No	Status
SSP_4274_26	● Gas	CHEAPER CONDITIONAL OPERATORS	Automated	L83 - L83	⚠ *Pending Fix*
SSP_4274_26	● Gas	CHEAPER CONDITIONAL OPERATORS	Automated	L245 - L245	⚠ *Pending Fix*
SSP_4274_24	● Gas	CHEAPER INEQUALITIES IN REQUIRE()	Automated	L476 - L476	⚠ *Pending Fix*
SSP_4274_1	● Gas	DEFINE CONSTRUCTOR AS PAYABLE	Automated	L22 - L24	⚠ *Pending Fix*
SSP_4274_22	● Gas	FUNCTION SHOULD RETURN STRUCT	Automated	L407 - L423	⚠ *Pending Fix*
SSP_4274_39	● Gas	LONG REQUIRE/REVERT STRINGS	Automated	L98 - L98	⚠ *Pending Fix*
SSP_4274_40	● Gas	LONG REQUIRE/REVERT STRINGS	Automated	L100 - L100	⚠ *Pending Fix*
SSP_4274_41	● Gas	LONG REQUIRE/REVERT STRINGS	Automated	L163 - L163	⚠ *Pending Fix*
SSP_4274_41	● Gas	LONG REQUIRE/REVERT STRINGS	Automated	L216 - L216	⚠ *Pending Fix*
SSP_4274_39	● Gas	LONG REQUIRE/REVERT STRINGS	Automated	L260 - L260	⚠ *Pending Fix*
SSP_4274_40	● Gas	LONG REQUIRE/REVERT STRINGS	Automated	L262 - L262	⚠ *Pending Fix*
SSP_4274_41	● Gas	LONG REQUIRE/REVERT STRINGS	Automated	L327 - L327	⚠ *Pending Fix*
SSP_4274_41	● Gas	LONG REQUIRE/REVERT STRINGS	Automated	L381 - L381	⚠ *Pending Fix*
SSP_4274_53	● Gas	LONG REQUIRE/REVERT STRINGS	Automated	L475 - L478	⚠ *Pending Fix*
SSP_4274_10	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L78 - L117	⚠ *Pending Fix*
SSP_4274_11	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L127 - L178	⚠ *Pending Fix*
SSP_4274_11	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L127 - L178	⚠ *Pending Fix*
SSP_4274_21	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L186 - L231	⚠ *Pending Fix*

Bug ID	Severity	Bug Type	Detection Method	Line No	Status
SSP_4274_21	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L186 - L231	⚠ *Pending Fix*
SSP_4274_28	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L240 - L280	⚠ *Pending Fix*
SSP_4274_28	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L240 - L280	⚠ *Pending Fix*
SSP_4274_29	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L290 - L342	⚠ *Pending Fix*
SSP_4274_29	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L290 - L342	⚠ *Pending Fix*
SSP_4274_42	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L350 - L396	⚠ *Pending Fix*
SSP_4274_42	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L350 - L396	⚠ *Pending Fix*
SSP_4274_49	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L432 - L452	⚠ *Pending Fix*
SSP_4274_37	● Gas	SUPERFLUOUS EVENT FIELDS	Automated	L110 - L116	⚠ *Pending Fix*
SSP_4274_38	● Gas	SUPERFLUOUS EVENT FIELDS	Automated	L273 - L279	⚠ *Pending Fix*
SSP_4274_50	● Gas	UNUSED IMPORTS	Automated	L4 - L4	⚠ *Pending Fix*
SSP_4274_51	● Gas	UNUSED IMPORTS	Automated	L5 - L5	⚠ *Pending Fix*

4. Vulnerability Details

Bug ID

SSP_4274_15

Bug Type

INCORRECT ACCESS CONTROL

Severity

● Critical

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L78 - L117

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L78 - L117

```

    function swapWithFeeBuy(
        address _tokenOut,
        uint256 _amountOutMin,
        address _to
    ) external payable ZeroAddress(_to) ZeroAmount(_amountOutMin) {
        require(msg.value > 0, "BC:Invalid ETH Amount");
        // Construct the token swap path
        address[] memory path;

        path = new address[](2);
        path[0] = WETH;
        path[1] = _tokenOut;

        (
            ,
            uint256 maintanierFee,
            uint256 platformFee,
            uint256 amountToSend
        ) = percentageCalculation(msg.value);
        (bool success, ) = maintanierAddress.call{value: maintanierFee}("");
        require(success, "ETH transfer failed To Maintainer");
        (success, ) = platformAddress.call{value: platformFee}("");
         require(success, "ETH transfer failed To Maintainer");

         uint256 _amountOut = IUniswapV2Router02(UNISWAP_V2_ROUTER)
             .swapExactETHForTokens{value: amountToSend}(
             _amountOutMin,
             path,
```

/BuyContract (1).sol L78 - L117

```
            .swapExactETHForTokens{value: amountToSend}(
            _amountOutMin,
            path,
            _to,
            block.timestamp
        )[0];

        emit TokensSwapped(
            WETH, // ETH address
            _tokenOut,
            amountToSend,
            _amountOut,
            _to
        );
    }

    /**
```

Description

Access control plays an important role in segregation of privileges in smart contracts and other applications. If this is misconfigured or not properly validated on sensitive functions, it may lead to loss of funds, tokens and in some cases compromise of the smart contract.

The contract BuyContract is importing an access control library @openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol but the function swapWithFeeBuy is missing the modifier onlyOwner.

Remediation

It is recommended to go through the contract and observe the functions that are lacking an access control modifier. If they contain sensitive administrative actions, it is advised to add a suitable modifier to the same

Bug ID

SSP_4274_16

Bug Type

INCORRECT ACCESS CONTROL

Severity

● Critical

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L127 - L178

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol

L127 - L178

```
    function swapWithFeeSell(
        address _tokenIn,
        uint256 _amountIn,
        uint256 _amountOutMin,
        address _to
    )
        external
        ZeroAddress(_to)
        ZeroAmount(_amountIn)
        ZeroAmount(_amountOutMin)
    {
        // Construct the token swap path
        address[] memory path = new address[](2);
        path[0] = _tokenIn;
        path[1] = WETH;

        IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
        IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);

        uint256 amount = IUniswapV2Router02(UNISWAP_V2_ROUTER)
            .swapExactTokensForETH(
                _amountIn,
                _amountOutMin,
                path,
                address(this),
                block.timestamp
            )[1];

```

/BuyContract (1).sol L127 - L178

```
                block.timestamp
            )[1];

        (
            ,
            uint256 maintanierFee,
            uint256 platformFee,
            uint256 amountToSend
        ) = percentageCalculation(amount);

        (bool success, ) = maintanierAddress.call{value: maintanierFee}("");
        require(success, "ETH transfer failed To Maintainer");
        success = false;
        (success, ) = platformAddress.call{value: platformFee}("");
        require(success, "ETH transfer failed To Platform");
        success = false;
        (success, ) = _to.call{value: amountToSend}("");
        require(success, "ETH transfer failed To User");

        emit TokensSwapped(
            _tokenIn,
            WETH, // ETH address
            _amountIn,
            amountToSend,
            _to
        );
    }

    /**
```

Description

Access control plays an important role in segregation of privileges in smart contracts and other applications. If this is misconfigured or not properly validated on sensitive functions, it may lead to loss of funds, tokens and in some cases compromise of the smart contract.

The contract BuyContract is importing an access control library @openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol but the function swapWithFeeSell is missing the modifier onlyOwner.

Remediation

It is recommended to go through the contract and observe the functions that are lacking an access control modifier. If they contain sensitive administrative actions, it is advised to add a suitable modifier to the same

Bug ID

SSP_4274_17

Bug Type

INCORRECT ACCESS CONTROL

Severity

● Critical

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L186 - L231

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol

L186 - L231

```
function quickSwapWithFeeSell(
    address _tokenIn,
    address _to
) external ZeroAddress(_to) {
    // Construct the token swap path
    uint256 _amountIn = IERC20(_tokenIn).balanceOf(msg.sender);
    address[] memory path = new address[](2);
    path[0] = _tokenIn;
    path[1] = WETH;

    IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
    IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);

    uint256 amount = IUniswapV2Router02(UNISWAP_V2_ROUTER)
        .swapExactTokensForETH(
            _amountIn,
            0,
            path,
            address(this),
            block.timestamp
        )[1];

    (
        ,
        uint256 maintanierFee,
        uint256 platformFee,
        uint256 amountToSend
    ) = percentageCalculation(amount);
```

/BuyContract (1).sol L186 - L231

```
            uint256 platformFee,
            uint256 amountToSend
        ) = percentageCalculation(amount);

        (bool success, ) = maintanierAddress.call{value: maintanierFee}("");
        require(success, "ETH transfer failed To Maintainer");
        success = false;
        (success, ) = platformAddress.call{value: platformFee}("");
        require(success, "ETH transfer failed To Platform");
        success = false;
        (success, ) = _to.call{value: amountToSend}("");
        require(success, "ETH transfer failed To User");

        emit TokensSwapped(
            _tokenIn,
            WETH, // ETH address
            _amountIn,
            amountToSend,
            _to
        );
    }

    /**
```

Description

Access control plays an important role in segregation of privileges in smart contracts and other applications. If this is misconfigured or not properly validated on sensitive functions, it may lead to loss of funds, tokens and in some cases compromise of the smart contract.

The contract BuyContract is importing an access control library @openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol but the function quickSwapWithFeeSell is missing the modifier onlyOwner.

Remediation

It is recommended to go through the contract and observe the functions that are lacking an access control modifier. If they contain sensitive administrative actions, it is advised to add a suitable modifier to the same

Bug ID

SSP_4274_18

Bug Type

INCORRECT ACCESS CONTROL

Severity

● Critical

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L240 - L280

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol

L240 - L280

```

function swapWithBuyTaxToken(
    address _tokenOut,
    uint256 _amountOutMin,
    address _to
) external payable ZeroAddress(_to) ZeroAmount(_amountOutMin) {
    require(msg.value > 0, "BC:Invalid ETH Amount");
    // Construct the token swap path
    address[] memory path;

    path = new address[](2);
    path[0] = WETH;
    path[1] = _tokenOut;

    (
        ,
        uint256 maintanierFee,
        uint256 platformFee,
        uint256 amountToSend
    ) = percentageCalculation(msg.value);
    (bool success, ) = maintanierAddress.call{value: maintanierFee}("");
    require(success, "ETH transfer failed To Maintainer");
    (success, ) = platformAddress.call{value: platformFee}("");
    require(success, "ETH transfer failed To Maintainer");

    IUniswapV2Router02(UNISWAP_V2_ROUTER)
        .swapExactETHForTokensSupportingFeeOnTransferTokens{
        value: amountToSend
    }(_amountOutMin, path, _to, block.timestamp);
```

/BuyContract (1).sol L240 - L280

```
            .swapExactETHForTokensSupportingFeeOnTransferTokens{
            value: amountToSend
        }(_amountOutMin, path, _to, block.timestamp);
        uint amount = IUniswapV2Router02(UNISWAP_V2_ROUTER).getAmountsOut(
            amountToSend,
            path
        )[0];

        emit TokensSwapped(
            WETH, // ETH address
            _tokenOut,
            amountToSend,
            amount,
            _to
        );
    }

    /**
```

Description

Access control plays an important role in segregation of privileges in smart contracts and other applications. If this is misconfigured or not properly validated on sensitive functions, it may lead to loss of funds, tokens and in some cases compromise of the smart contract.

The contract BuyContract is importing an access control library @openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol but the function swapWithBuyTaxToken is missing the modifier onlyOwner.

Remediation

It is recommended to go through the contract and observe the functions that are lacking an access control modifier. If they contain sensitive administrative actions, it is advised to add a suitable modifier to the same

Bug ID

SSP_4274_19

Bug Type

INCORRECT ACCESS CONTROL

Severity

● Critical

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L290 - L342

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol

L290 - L342

```
function swapWithSellTaxToken(
    address _tokenIn,
    uint256 _amountIn,
    uint256 _amountOutMin,
    address _to,
    uint _afterTax
)
    external
    ZeroAddress(_to)
    ZeroAmount(_amountIn)
    ZeroAmount(_amountOutMin)
{
    // Construct the token swap path
    address[] memory path = new address[](2);
    path[0] = _tokenIn;
    path[1] = WETH;

    IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
    IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);

    IUniswapV2Router02(UNISWAP_V2_ROUTER)
        .swapExactTokensForETHSupportingFeeOnTransferTokens(
            IERC20(_tokenIn).balanceOf(address(this)),
            _amountOutMin,
            path,
            address(this),
            block.timestamp
        );
```

/BuyContract (1).sol L290 - L342

```
                address(this),
                block.timestamp
            );

        (
            ,
            uint256 maintanierFee,
            uint256 platformFee,
            uint256 amountToSend
        ) = percentageCalculation(_afterTax);

        (bool success, ) = maintanierAddress.call{value: maintanierFee}("");
        require(success, "ETH transfer failed To Maintainer");
        success = false;
        (success, ) = platformAddress.call{value: platformFee}("");
        require(success, "ETH transfer failed To Platform");
        success = false;
        (success, ) = _to.call{value: amountToSend}("");
        require(success, "ETH transfer failed To User");

        emit TokensSwapped(
            _tokenIn,
            WETH, // ETH address
            _amountIn,
            amountToSend,
            _to
        );
    }

    /**
```

Description

Access control plays an important role in segregation of privileges in smart contracts and other applications. If this is misconfigured or not properly validated on sensitive functions, it may lead to loss of funds, tokens and in some cases compromise of the smart contract.

The contract BuyContract is importing an access control library @openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol but the function swapWithSellTaxToken is missing the modifier onlyOwner.

Remediation

It is recommended to go through the contract and observe the functions that are lacking an access control modifier. If they contain sensitive administrative actions, it is advised to add a suitable modifier to the same

Bug ID

SSP_4274_20

Bug Type

INCORRECT ACCESS CONTROL

Severity

● Critical

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L350 - L396

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L350 - L396

```

    function quickSwapWithSellTaxToken(
        address _tokenIn,
        address _to,
        uint _afterTax
    ) external ZeroAddress(_to) {
        // Construct the token swap path
        uint256 _amountIn = IERC20(_tokenIn).balanceOf(msg.sender);
        address[] memory path = new address[](2);
        path[0] = _tokenIn;
        path[1] = WETH;

        IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
        IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);

        IUniswapV2Router02(UNISWAP_V2_ROUTER)
            .swapExactTokensForETHSupportingFeeOnTransferTokens(
                IERC20(_tokenIn).balanceOf(address(this)),
                0,
                path,
                address(this),
                block.timestamp
            );

        (
            ,
            uint256 maintanierFee,
            uint256 platformFee,
            uint256 amountToSend
```

/BuyContract (1).sol L350 - L396

```
            uint256 maintanierFee,
            uint256 platformFee,
            uint256 amountToSend
        ) = percentageCalculation(_afterTax);

        (bool success, ) = maintanierAddress.call{value: maintanierFee}("");
        require(success, "ETH transfer failed To Maintainer");
        success = false;
        (success, ) = platformAddress.call{value: platformFee}("");
        require(success, "ETH transfer failed To Platform");
        success = false;
        (success, ) = _to.call{value: amountToSend}("");
        require(success, "ETH transfer failed To User");

        emit TokensSwapped(
            _tokenIn,
            WETH, // ETH address
            _amountIn,
            amountToSend,
            _to
        );
    }

    /**
```

Description

Access control plays an important role in segregation of privileges in smart contracts and other applications. If this is misconfigured or not properly validated on sensitive functions, it may lead to loss of funds, tokens and in some cases compromise of the smart contract.

The contract BuyContract is importing an access control library @openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol but the function quickSwapWithSellTaxToken is missing the modifier onlyOwner.

Remediation

It is recommended to go through the contract and observe the functions that are lacking an access control modifier. If they contain sensitive administrative actions, it is advised to add a suitable modifier to the same

Bug ID

SSP_4274_30

Bug Type

REENTRANCY

Severity

● High

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L78 - L117

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L78 - L117

```

    function swapWithFeeBuy(
        address _tokenOut,
        uint256 _amountOutMin,
        address _to
    ) external payable ZeroAddress(_to) ZeroAmount(_amountOutMin) {
        require(msg.value > 0, "BC:Invalid ETH Amount");
        // Construct the token swap path
        address[] memory path;

        path = new address[](2);
        path[0] = WETH;
        path[1] = _tokenOut;

        (
            ,
            uint256 maintanierFee,
            uint256 platformFee,
            uint256 amountToSend
        ) = percentageCalculation(msg.value);
        (bool success, ) = maintanierAddress.call{value: maintanierFee}("");
        require(success, "ETH transfer failed To Maintainer");
        (success, ) = platformAddress.call{value: platformFee}("");
         require(success, "ETH transfer failed To Maintainer");

         uint256 _amountOut = IUniswapV2Router02(UNISWAP_V2_ROUTER)
             .swapExactETHForTokens{value: amountToSend}(
             _amountOutMin,
             path,
```

/BuyContract (1).sol L78 - L117

```
            .swapExactETHForTokens{value: amountToSend}(
            _amountOutMin,
            path,
            _to,
            block.timestamp
        )[0];

        emit TokensSwapped(
            WETH, // ETH address
            _tokenOut,
            amountToSend,
            _amountOut,
            _to
        );
    }

    /**
```

Description

In a Re-entrancy attack, a malicious contract calls back into the calling contract before the first invocation of the function is finished. This may cause the different invocations of the function to interact in undesirable ways, especially in cases where the function is updating state variables after the external calls.
This may lead to loss of funds, improper value updates, token loss, etc.

Remediation

It is recommended to add a [Re-entrancy Guard] to the functions making external calls. The functions should use a Checks-Effects-Interactions pattern. The external calls should be executed at the end of the function and all the state-changing must happen before the call.

Bug ID	Bug Type	
SSP_4274_31	**REENTRANCY**	
Severity	Action Taken	Detection Method
● High	⚠ *Pending Fix*	Automated

Line No.	File Location
L127 - L178	/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L127 - L178

```

function swapWithFeeSell(
    address _tokenIn,
    uint256 _amountIn,
    uint256 _amountOutMin,
    address _to
)
    external
    ZeroAddress(_to)
    ZeroAmount(_amountIn)
    ZeroAmount(_amountOutMin)
{
    // Construct the token swap path
    address[] memory path = new address[](2);
    path[0] = _tokenIn;
    path[1] = WETH;

    IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
    IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);

    uint256 amount = IUniswapV2Router02(UNISWAP_V2_ROUTER)
        .swapExactTokensForETH(
            _amountIn,
            _amountOutMin,
            path,
            address(this),
            block.timestamp
        )[1];

```

/BuyContract (1).sol L127 - L178

```
                block.timestamp
            )[1];

        (
            ,
            uint256 maintanierFee,
            uint256 platformFee,
            uint256 amountToSend
        ) = percentageCalculation(amount);

        (bool success, ) = maintanierAddress.call{value: maintanierFee}("");
        require(success, "ETH transfer failed To Maintainer");
        success = false;
        (success, ) = platformAddress.call{value: platformFee}("");
        require(success, "ETH transfer failed To Platform");
        success = false;
        (success, ) = _to.call{value: amountToSend}("");
        require(success, "ETH transfer failed To User");

        emit TokensSwapped(
            _tokenIn,
            WETH, // ETH address
            _amountIn,
            amountToSend,
            _to
        );
    }

    /**
```

Description

In a Re-entrancy attack, a malicious contract calls back into the calling contract before the first invocation of the function is finished. This may cause the different invocations of the function to interact in undesirable ways, especially in cases where the function is updating state variables after the external calls.
This may lead to loss of funds, improper value updates, token loss, etc.

Remediation

It is recommended to add a [Re-entrancy Guard] to the functions making external calls. The functions should use a Checks-Effects-Interactions pattern. The external calls should be executed at the end of the function and all the state-changing must happen before the call.

Bug ID	Bug Type	
SSP_4274_32	**REENTRANCY**	
Severity	Action Take	Detection Method
● High	⚠ *Pending Fix*	Automated

Line No.	File Location
L186 - L231	/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L186 - L231

```
    function quickSwapWithFeeSell(
        address _tokenIn,
        address _to
    ) external ZeroAddress(_to) {
        // Construct the token swap path
        uint256 _amountIn = IERC20(_tokenIn).balanceOf(msg.sender);
        address[] memory path = new address[](2);
        path[0] = _tokenIn;
        path[1] = WETH;

        IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
        IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);

        uint256 amount = IUniswapV2Router02(UNISWAP_V2_ROUTER)
            .swapExactTokensForETH(
                _amountIn,
                0,
                path,
                address(this),
                block.timestamp
            )[1];

        (
            ,
            uint256 maintanierFee,
            uint256 platformFee,
            uint256 amountToSend
        ) = percentageCalculation(amount);
```

/BuyContract (1).sol L186 - L231

```
            uint256 platformFee,
            uint256 amountToSend
        ) = percentageCalculation(amount);

        (bool success, ) = maintanierAddress.call{value: maintanierFee}("");
        require(success, "ETH transfer failed To Maintainer");
        success = false;
        (success, ) = platformAddress.call{value: platformFee}("");
        require(success, "ETH transfer failed To Platform");
        success = false;
        (success, ) = _to.call{value: amountToSend}("");
        require(success, "ETH transfer failed To User");

        emit TokensSwapped(
            _tokenIn,
            WETH, // ETH address
            _amountIn,
            amountToSend,
            _to
        );
    }

    /**
```

Description

In a Re-entrancy attack, a malicious contract calls back into the calling contract before the first invocation of the function is finished. This may cause the different invocations of the function to interact in undesirable ways, especially in cases where the function is updating state variables after the external calls.
This may lead to loss of funds, improper value updates, token loss, etc.

Remediation

It is recommended to add a [Re-entrancy Guard] to the functions making external calls. The functions should use a Checks-Effects-Interactions pattern. The external calls should be executed at the end of the function and all the state-changing must happen before the call.

Bug ID

SSP_4274_33

Bug Type

REENTRANCY

Severity

● High

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L240 - L280

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L240 - L280

```

    function swapWithBuyTaxToken(
        address _tokenOut,
        uint256 _amountOutMin,
        address _to
    ) external payable ZeroAddress(_to) ZeroAmount(_amountOutMin) {
        require(msg.value > 0, "BC:Invalid ETH Amount");
        // Construct the token swap path
        address[] memory path;

        path = new address[](2);
        path[0] = WETH;
        path[1] = _tokenOut;

        (
            ,
            uint256 maintanierFee,
            uint256 platformFee,
            uint256 amountToSend
        ) = percentageCalculation(msg.value);
        (bool success, ) = maintanierAddress.call{value: maintanierFee}("");
        require(success, "ETH transfer failed To Maintainer");
        (success, ) = platformAddress.call{value: platformFee}("");
        require(success, "ETH transfer failed To Maintainer");

        IUniswapV2Router02(UNISWAP_V2_ROUTER)
            .swapExactETHForTokensSupportingFeeOnTransferTokens{
            value: amountToSend
        }(_amountOutMin, path, _to, block.timestamp);
```

/BuyContract (1).sol L240 - L280

```
            .swapExactETHForTokensSupportingFeeOnTransferTokens{
            value: amountToSend
        }(_amountOutMin, path, _to, block.timestamp);
        uint amount = IUniswapV2Router02(UNISWAP_V2_ROUTER).getAmountsOut(
            amountToSend,
            path
        )[0];

        emit TokensSwapped(
            WETH, // ETH address
            _tokenOut,
            amountToSend,
            amount,
            _to
        );
    }

    /**
```

Description

In a Re-entrancy attack, a malicious contract calls back into the calling contract before the first invocation of the function is finished. This may cause the different invocations of the function to interact in undesirable ways, especially in cases where the function is updating state variables after the external calls.
This may lead to loss of funds, improper value updates, token loss, etc.

Remediation

It is recommended to add a [Re-entrancy Guard] to the functions making external calls. The functions should use a Checks-Effects-Interactions pattern. The external calls should be executed at the end of the function and all the state-changing must happen before the call.

Bug ID

SSP_4274_34

Bug Type

REENTRANCY

Severity

● High

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L290 - L342

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol

L290 - L342

```
    function swapWithSellTaxToken(
        address _tokenIn,
        uint256 _amountIn,
        uint256 _amountOutMin,
        address _to,
        uint _afterTax
    )
        external
        ZeroAddress(_to)
        ZeroAmount(_amountIn)
        ZeroAmount(_amountOutMin)
    {
        // Construct the token swap path
        address[] memory path = new address[](2);
        path[0] = _tokenIn;
        path[1] = WETH;

        IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
        IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);

        IUniswapV2Router02(UNISWAP_V2_ROUTER)
            .swapExactTokensForETHSupportingFeeOnTransferTokens(
                IERC20(_tokenIn).balanceOf(address(this)),
                _amountOutMin,
                path,
                address(this),
                block.timestamp
            );
```

/BuyContract (1).sol L290 - L342

```
                address(this),
                block.timestamp
            );

        (
            ,
            uint256 maintanierFee,
            uint256 platformFee,
            uint256 amountToSend
        ) = percentageCalculation(_afterTax);

        (bool success, ) = maintanierAddress.call{value: maintanierFee}("");
        require(success, "ETH transfer failed To Maintainer");
        success = false;
        (success, ) = platformAddress.call{value: platformFee}("");
        require(success, "ETH transfer failed To Platform");
        success = false;
        (success, ) = _to.call{value: amountToSend}("");
        require(success, "ETH transfer failed To User");

        emit TokensSwapped(
            _tokenIn,
            WETH, // ETH address
            _amountIn,
            amountToSend,
            _to
        );
    }

    /**
```

Description

In a Re-entrancy attack, a malicious contract calls back into the calling contract before the first invocation of the function is finished. This may cause the different invocations of the function to interact in undesirable ways, especially in cases where the function is updating state variables after the external calls.
This may lead to loss of funds, improper value updates, token loss, etc.

Remediation

It is recommended to add a [Re-entrancy Guard] to the functions making external calls. The functions should use a Checks-Effects-Interactions pattern. The external calls should be executed at the end of the function and all the state-changing must happen before the call.

Bug ID	Bug Type	
SSP_4274_35	**REENTRANCY**	
Severity	Action Taken	Detection Method
● High	⚠ *Pending Fix*	Automated

Line No.	File Location
L350 - L396	/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L350 - L396

```
349
350     function quickSwapWithSellTaxToken(
351         address _tokenIn,
352         address _to,
353         uint _afterTax
354     ) external ZeroAddress(_to) {
355         // Construct the token swap path
356         uint256 _amountIn = IERC20(_tokenIn).balanceOf(msg.sender);
357         address[] memory path = new address[](2);
358         path[0] = _tokenIn;
359         path[1] = WETH;
360
361         IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
362         IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);
363
364         IUniswapV2Router02(UNISWAP_V2_ROUTER)
365             .swapExactTokensForETHSupportingFeeOnTransferTokens(
366                 IERC20(_tokenIn).balanceOf(address(this)),
367                 0,
368                 path,
369                 address(this),
370                 block.timestamp
371             );
372
373         (
374             ,
375             uint256 maintanierFee,
376             uint256 platformFee,
377             uint256 amountToSend
```

/BuyContract (1).sol L350 - L396

```
            uint256 maintanierFee,
            uint256 platformFee,
            uint256 amountToSend
        ) = percentageCalculation(_afterTax);

        (bool success, ) = maintanierAddress.call{value: maintanierFee}("");
        require(success, "ETH transfer failed To Maintainer");
        success = false;
        (success, ) = platformAddress.call{value: platformFee}("");
        require(success, "ETH transfer failed To Platform");
        success = false;
        (success, ) = _to.call{value: amountToSend}("");
        require(success, "ETH transfer failed To User");

        emit TokensSwapped(
            _tokenIn,
            WETH, // ETH address
            _amountIn,
            amountToSend,
            _to
        );
    }

    /**
```

Description

In a Re-entrancy attack, a malicious contract calls back into the calling contract before the first invocation of the function is finished. This may cause the different invocations of the function to interact in undesirable ways, especially in cases where the function is updating state variables after the external calls.
This may lead to loss of funds, improper value updates, token loss, etc.

Remediation

It is recommended to add a [Re-entrancy Guard] to the functions making external calls. The functions should use a Checks-Effects-Interactions pattern. The external calls should be executed at the end of the function and all the state-changing must happen before the call.

Bug ID	Bug Type	
SSP_4274_9	**UNCHECKED TRANSFER**	
Severity	Action Táken	Detection Method
● High	⚠ *Pending Fix*	Automated

Line No.	File Location
L143 - L143	/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L143 - L143

```
142
143         IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
144         IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);
145
```

Description

Some tokens do not revert the transaction when the transfer or transferFrom fails and returns False. Hence we must check the return value after calling the `transfer` or `transferFrom` function.

Remediation

Use OpenZeppelin SafeERC20's `safetransfer` and `safetransferFrom` functions.

Bug ID	Bug Type	
SSP_4274_9	**UNCHECKED TRANSFER**	
Severity	Action Taken	Detection Method
● High	⚠ *Pending Fix*	Automated

Line No.	File Location
L196 - L196	/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L196 - L196

```
195
196         IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
197         IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);
198
```

Description

Some tokens do not revert the transaction when the transfer or transferFrom fails and returns False. Hence we must ch eck the return value after calling the `transfer` or `transferFrom` function.

Remediation

Use OpenZeppelin SafeERC20's `safetransfer` and `safetransferFrom` functions.

Bug ID

SSP_4274_9

Bug Type

UNCHECKED TRANSFER

Severity

● High

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L307 - L307

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L307 - L307

```
306
307         IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
308         IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);
309
```

Description

Some tokens do not revert the transaction when the transfer or transferFrom fails and returns False. Hence we must check the return value after calling the `transfer` or `transferFrom` function.

Remediation

Use OpenZeppelin SafeERC20's `safetransfer` and `safetransferFrom` functions.

Bug ID	Bug Type	
SSP_4274_9	**UNCHECKED TRANSFER**	
Severity	Action Taken	Detection Method
● High	⚠ *Pending Fix*	Automated

Line No.	File Location
L361 - L361	/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L361 - L361

```

        IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
        IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);

```

Description

Some tokens do not revert the transaction when the transfer or transferFrom fails and returns False. Hence we must check the return value after calling the `transfer` or `transferFrom` function.

Remediation

Use OpenZeppelin SafeERC20's `safetransfer` and `safetransferFrom` functions.

Bug ID

SSP_4274_43

Bug Type

EVENT BASED REENTRANCY

Severity

● Low

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L78 - L117

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L78 - L117

```
    function swapWithFeeBuy(
        address _tokenOut,
        uint256 _amountOutMin,
        address _to
    ) external payable ZeroAddress(_to) ZeroAmount(_amountOutMin) {
        require(msg.value > 0, "BC:Invalid ETH Amount");
        // Construct the token swap path
        address[] memory path;

        path = new address[](2);
        path[0] = WETH;
        path[1] = _tokenOut;

        (
            ,
            uint256 maintanierFee,
            uint256 platformFee,
            uint256 amountToSend
        ) = percentageCalculation(msg.value);
        (bool success, ) = maintanierAddress.call{value: maintanierFee}("");
        require(success, "ETH transfer failed To Maintainer");
        (success, ) = platformAddress.call{value: platformFee}("");
         require(success, "ETH transfer failed To Maintainer");

         uint256 _amountOut = IUniswapV2Router02(UNISWAP_V2_ROUTER)
             .swapExactETHForTokens{value: amountToSend}(
             _amountOutMin,
             path,
```

/BuyContract (1).sol L78 - L117

```
            .swapExactETHForTokens{value: amountToSend}(
            _amountOutMin,
            path,
            _to,
            block.timestamp
        )[0];

        emit TokensSwapped(
            WETH, // ETH address
            _tokenOut,
            amountToSend,
            _amountOut,
            _to
        );
    }

    /**
```

Description

In a Re-entrancy attack, a malicious contract calls back into the calling contract before the first invocation of the functi on is finished. This may cause the different invocations of the function to interact in undesirable ways, especially in cas es where the function is updating state variables after the external calls.
In the case of event-based Re-entrancy attacks, events are emitted after an external call leading to missing event call s.

Remediation

It is recommended to add a [Re-entrancy Guard] to the functions making external calls. The functions should use a Che cks-Effects-Interactions pattern. The external calls should be executed at the end of the function and all the state-cha nging and event emits must happen before the call.

Bug ID

SSP_4274_44

Bug Type

EVENT BASED REENTRANCY

Severity

● Low

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L127 - L178

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L127 - L178

```

function swapWithFeeSell(
    address _tokenIn,
    uint256 _amountIn,
    uint256 _amountOutMin,
    address _to
)
    external
    ZeroAddress(_to)
    ZeroAmount(_amountIn)
    ZeroAmount(_amountOutMin)
{
    // Construct the token swap path
    address[] memory path = new address[](2);
    path[0] = _tokenIn;
    path[1] = WETH;

    IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
    IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);

    uint256 amount = IUniswapV2Router02(UNISWAP_V2_ROUTER)
        .swapExactTokensForETH(
            _amountIn,
            _amountOutMin,
            path,
            address(this),
            block.timestamp
        )[1];

```

/BuyContract (1).sol L127 - L178

```
                block.timestamp
            )[1];

        (
            ,
            uint256 maintanierFee,
            uint256 platformFee,
            uint256 amountToSend
        ) = percentageCalculation(amount);

        (bool success, ) = maintanierAddress.call{value: maintanierFee}("");
        require(success, "ETH transfer failed To Maintainer");
        success = false;
        (success, ) = platformAddress.call{value: platformFee}("");
        require(success, "ETH transfer failed To Platform");
        success = false;
        (success, ) = _to.call{value: amountToSend}("");
        require(success, "ETH transfer failed To User");

        emit TokensSwapped(
            _tokenIn,
            WETH, // ETH address
            _amountIn,
            amountToSend,
            _to
        );
    }

    /**
```

Description

In a Re-entrancy attack, a malicious contract calls back into the calling contract before the first invocation of the function is finished. This may cause the different invocations of the function to interact in undesirable ways, especially in cases where the function is updating state variables after the external calls.
In the case of event-based Re-entrancy attacks, events are emitted after an external call leading to missing event calls.

Remediation

It is recommended to add a [Re-entrancy Guard] to the functions making external calls. The functions should use a Checks-Effects-Interactions pattern. The external calls should be executed at the end of the function and all the state-changing and event emits must happen before the call.

Bug ID

SSP_4274_45

Bug Type

EVENT BASED REENTRANCY

Severity

● Low

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L186 - L231

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L186 - L231

```

    function quickSwapWithFeeSell(
        address _tokenIn,
        address _to
    ) external ZeroAddress(_to) {
        // Construct the token swap path
        uint256 _amountIn = IERC20(_tokenIn).balanceOf(msg.sender);
        address[] memory path = new address[](2);
        path[0] = _tokenIn;
        path[1] = WETH;

        IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
        IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);

        uint256 amount = IUniswapV2Router02(UNISWAP_V2_ROUTER)
            .swapExactTokensForETH(
                _amountIn,
                0,
                path,
                address(this),
                block.timestamp
            )[1];

        (
            ,
            uint256 maintanierFee,
            uint256 platformFee,
            uint256 amountToSend
        ) = percentageCalculation(amount);
```

/BuyContract (1).sol L186 - L231

```
            uint256 platformFee,
            uint256 amountToSend
        ) = percentageCalculation(amount);

        (bool success, ) = maintanierAddress.call{value: maintanierFee}("");
        require(success, "ETH transfer failed To Maintainer");
        success = false;
        (success, ) = platformAddress.call{value: platformFee}("");
        require(success, "ETH transfer failed To Platform");
        success = false;
        (success, ) = _to.call{value: amountToSend}("");
        require(success, "ETH transfer failed To User");

        emit TokensSwapped(
            _tokenIn,
            WETH, // ETH address
            _amountIn,
            amountToSend,
            _to
        );
    }

    /**
```

Description

In a Re-entrancy attack, a malicious contract calls back into the calling contract before the first invocation of the function is finished. This may cause the different invocations of the function to interact in undesirable ways, especially in cases where the function is updating state variables after the external calls.
In the case of event-based Re-entrancy attacks, events are emitted after an external call leading to missing event calls.

Remediation

It is recommended to add a [Re-entrancy Guard] to the functions making external calls. The functions should use a Checks-Effects-Interactions pattern. The external calls should be executed at the end of the function and all the state-changing and event emits must happen before the call.

Bug ID

SSP_4274_46

Bug Type

EVENT BASED REENTRANCY

Severity

● Low

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L240 - L280

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol

L240 - L280

```

function swapWithBuyTaxToken(
    address _tokenOut,
    uint256 _amountOutMin,
    address _to
) external payable ZeroAddress(_to) ZeroAmount(_amountOutMin) {
    require(msg.value > 0, "BC:Invalid ETH Amount");
    // Construct the token swap path
    address[] memory path;

    path = new address[](2);
    path[0] = WETH;
    path[1] = _tokenOut;

    (
        ,
        uint256 maintanierFee,
        uint256 platformFee,
        uint256 amountToSend
    ) = percentageCalculation(msg.value);
    (bool success, ) = maintanierAddress.call{value: maintanierFee}("");
    require(success, "ETH transfer failed To Maintainer");
    (success, ) = platformAddress.call{value: platformFee}("");
    require(success, "ETH transfer failed To Maintainer");

    IUniswapV2Router02(UNISWAP_V2_ROUTER)
        .swapExactETHForTokensSupportingFeeOnTransferTokens{
        value: amountToSend
    }(_amountOutMin, path, _to, block.timestamp);
```

/BuyContract (1).sol L240 - L280

```
            .swapExactETHForTokensSupportingFeeOnTransferTokens{
            value: amountToSend
        }(_amountOutMin, path, _to, block.timestamp);
        uint amount = IUniswapV2Router02(UNISWAP_V2_ROUTER).getAmountsOut(
            amountToSend,
            path
        )[0];

        emit TokensSwapped(
            WETH, // ETH address
            _tokenOut,
            amountToSend,
            amount,
            _to
        );
    }

    /**
```

Description

In a Re-entrancy attack, a malicious contract calls back into the calling contract before the first invocation of the function is finished. This may cause the different invocations of the function to interact in undesirable ways, especially in cases where the function is updating state variables after the external calls.
In the case of event-based Re-entrancy attacks, events are emitted after an external call leading to missing event calls.

Remediation

It is recommended to add a [Re-entrancy Guard] to the functions making external calls. The functions should use a Checks-Effects-Interactions pattern. The external calls should be executed at the end of the function and all the state-changing and event emits must happen before the call.

Bug ID

SSP_4274_47

Bug Type

EVENT BASED REENTRANCY

Severity

● Low

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L290 - L342

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L290 - L342

```

function swapWithSellTaxToken(
    address _tokenIn,
    uint256 _amountIn,
    uint256 _amountOutMin,
    address _to,
    uint _afterTax
)
    external
    ZeroAddress(_to)
    ZeroAmount(_amountIn)
    ZeroAmount(_amountOutMin)
{
    // Construct the token swap path
    address[] memory path = new address[](2);
    path[0] = _tokenIn;
    path[1] = WETH;

    IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
    IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);

    IUniswapV2Router02(UNISWAP_V2_ROUTER)
        .swapExactTokensForETHSupportingFeeOnTransferTokens(
            IERC20(_tokenIn).balanceOf(address(this)),
            _amountOutMin,
            path,
            address(this),
            block.timestamp
        );
```

/BuyContract (1).sol L290 - L342

```
                address(this),
                block.timestamp
            );

        (
            ,
            uint256 maintanierFee,
            uint256 platformFee,
            uint256 amountToSend
        ) = percentageCalculation(_afterTax);

        (bool success, ) = maintanierAddress.call{value: maintanierFee}("");
        require(success, "ETH transfer failed To Maintainer");
        success = false;
        (success, ) = platformAddress.call{value: platformFee}("");
        require(success, "ETH transfer failed To Platform");
        success = false;
        (success, ) = _to.call{value: amountToSend}("");
        require(success, "ETH transfer failed To User");

        emit TokensSwapped(
            _tokenIn,
            WETH, // ETH address
            _amountIn,
            amountToSend,
            _to
        );
    }

    /**
```

Description

In a Re-entrancy attack, a malicious contract calls back into the calling contract before the first invocation of the function is finished. This may cause the different invocations of the function to interact in undesirable ways, especially in cases where the function is updating state variables after the external calls.

In the case of event-based Re-entrancy attacks, events are emitted after an external call leading to missing event calls.

Remediation

It is recommended to add a [Re-entrancy Guard] to the functions making external calls. The functions should use a Checks-Effects-Interactions pattern. The external calls should be executed at the end of the function and all the state-changing and event emits must happen before the call.

Bug ID	Bug Type	
SSP_4274_48	**EVENT BASED REENTRANCY**	
Severity	Action Taken	Detection Method
● Low	⚠ *Pending Fix*	Automated

Line No.	File Location
L350 - L396	/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L350 - L396

```
    function quickSwapWithSellTaxToken(
        address _tokenIn,
        address _to,
        uint _afterTax
    ) external ZeroAddress(_to) {
        // Construct the token swap path
        uint256 _amountIn = IERC20(_tokenIn).balanceOf(msg.sender);
        address[] memory path = new address[](2);
        path[0] = _tokenIn;
        path[1] = WETH;

        IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
        IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);

        IUniswapV2Router02(UNISWAP_V2_ROUTER)
            .swapExactTokensForETHSupportingFeeOnTransferTokens(
                IERC20(_tokenIn).balanceOf(address(this)),
                0,
                path,
                address(this),
                block.timestamp
            );

        (
            ,
            uint256 maintanierFee,
            uint256 platformFee,
            uint256 amountToSend
```

/BuyContract (1).sol L350 - L396

```
            uint256 maintanierFee,
            uint256 platformFee,
            uint256 amountToSend
        ) = percentageCalculation(_afterTax);

        (bool success, ) = maintanierAddress.call{value: maintanierFee}("");
        require(success, "ETH transfer failed To Maintainer");
        success = false;
        (success, ) = platformAddress.call{value: platformFee}("");
        require(success, "ETH transfer failed To Platform");
        success = false;
        (success, ) = _to.call{value: amountToSend}("");
        require(success, "ETH transfer failed To User");

        emit TokensSwapped(
            _tokenIn,
            WETH, // ETH address
            _amountIn,
            amountToSend,
            _to
        );
    }

    /**
```

Description

In a Re-entrancy attack, a malicious contract calls back into the calling contract before the first invocation of the function is finished. This may cause the different invocations of the function to interact in undesirable ways, especially in cases where the function is updating state variables after the external calls.
In the case of event-based Re-entrancy attacks, events are emitted after an external call leading to missing event calls.

Remediation

It is recommended to add a [Re-entrancy Guard] to the functions making external calls. The functions should use a Checks-Effects-Interactions pattern. The external calls should be executed at the end of the function and all the state-changing and event emits must happen before the call.

Bug ID	Bug Type	
SSP_4274_23	**USE OF FLOATING PRAGMA**	
Severity	Action Taken	Detection Method
● Low	⚠ *Pending Fix*	Automated
Line No.	File Location	
L2 - L2	/BuyContract (1).sol	

Affected Code

/BuyContract (1).sol L2 - L2

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.7;
import "uniswap-v2-contract/contracts/uniswap-v2-periphery/interfaces/IUniswapV2Router02.sol";
import "uniswap-v2-contract/contracts/uniswap-v2-core/interfaces/IUniswapV2Factory.sol";
```

Description

Solidity source files indicate the versions of the compiler they can be compiled with using a pragma directive at the top of the solidity file. This can either be a floating pragma or a specific compiler version.
The contract was found to be using a floating pragma which is not considered safe as it can be compiled with all the versions described.
The following affected files were found to be using floating pragma:
`['/BuyContract (1).sol'] - ^0.8.7`

Remediation

It is recommended to use a fixed pragma version, as future compiler versions may handle certain language constructions in a way the developer did not foresee.
Using a floating pragma may introduce several vulnerabilities if compiled with an older version.
The developers should always use the exact Solidity compiler version when designing their contracts as it may break the changes in the future.
Instead of `^0.8.7` use `pragma solidity v0.8.23`, which is a stable and recommended version right now.

Bug ID	Bug Type	
SSP_4274_12	**MISSING EVENTS**	
Severity	Action Taken	Detection Method
● Low	⚠ *Pending Fix*	Automated

Line No.	File Location
L454 - L458	/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L454 - L458

```
453
454        function setPlatformAddress(
455            address _account
456        ) external onlyOwner ZeroAddress(_account) {
457            platformAddress = _account;
458        }
459
460        function setMaintainerAddress(
```

Description

Events are inheritable members of contracts. When you call them, they cause the arguments to be stored in the transaction's log—a special data structure in the blockchain.
These logs are associated with the address of the contract which can then be used by developers and auditors to keep track of the transactions.
The contract BuyContract was found to be missing these events on the function setPlatformAddress which would make it difficult or impossible to track these transactions off-chain.

Remediation

Consider emitting events for the functions mentioned above. It is also recommended to have the addresses indexed.

Bug ID	Bug Type	
SSP_4274_13	**MISSING EVENTS**	
Severity	Action Taken	Detection Method
● Low	⚠ *Pending Fix*	Automated

Line No.	File Location
L460 - L464	/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L460 - L464

```
459
460        function setMaintainerAddress(
461            address _account
462        ) external onlyOwner ZeroAddress(_account) {
463            maintanierAddress = _account;
464        }
465
466        receive() external payable {
```

Description

Events are inheritable members of contracts. When you call them, they cause the arguments to be stored in the transaction's log—a special data structure in the blockchain.
These logs are associated with the address of the contract which can then be used by developers and auditors to keep track of the transactions.
The contract BuyContract was found to be missing these events on the function setMaintainerAddress which would make it difficult or impossible to track these transactions off-chain.

Remediation

Consider emitting events for the functions mentioned above. It is also recommended to have the addresses indexed.

Bug ID

SSP_4274_14

Bug Type

MISSING EVENTS

Severity

● Low

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L470 - L481

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L470 - L481

```

    function withdrawEther(
        address payable recipient,
        uint256 amount
    ) external onlyOwner {
        require(recipient != address(0), "Invalid recipient address");
        require(
            address(this).balance >= amount,
            "Insufficient balance in the contract"
        );

        recipient.transfer(amount);
    }
}

```

Description

Events are inheritable members of contracts. When you call them, they cause the arguments to be stored in the transaction's log—a special data structure in the blockchain.
These logs are associated with the address of the contract which can then be used by developers and auditors to keep track of the transactions.
The contract BuyContract was found to be missing these events on the function withdrawEther which would make it difficult or impossible to track these transactions off-chain.

Remediation

Consider emitting events for the functions mentioned above. It is also recommended to have the addresses indexed.

Bug ID	Bug Type	
SSP_4274_25	**OUTDATED COMPILER VERSION**	
Severity	Action Taken	Detection Method
● Low	⚠ *Pending Fix*	Automated

Line No.	File Location
L2 - L2	/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L2 - L2

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.7;
import "uniswap-v2-contract/contracts/uniswap-v2-periphery/interfaces/IUniswapV2Router02.sol";
import "uniswap-v2-contract/contracts/uniswap-v2-core/interfaces/IUniswapV2Factory.sol";
```

Description

Using an outdated compiler version can be problematic especially if there are publicly disclosed bugs and issues that affect the current compiler version.
The following outdated versions were detected:
`['/BuyContract (1).sol']` - `^0.8.7`

Remediation

It is recommended to use a recent version of the Solidity compiler that should not be the most recent version, and it should not be an outdated version as well. Using very old versions of Solidity prevents the benefits of bug fixes and newer security checks. Consider using the solidity version `v0.8.23`, which patches most solidity vulnerabilities.

Bug ID

SSP_4274_5

Bug Type

USE OWNABLE2STEP

Severity

Low

Action Taken

Pending Fix

Detection Method

Automated

Line No.

L10 - L10

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L10 - L10

```

contract BuyContract is OwnableUpgradeable {
    // Address of the Uniswap v2 router
    address public UNISWAP_V2_ROUTER;
```

Description

`Ownable2Step` is safer than `Ownable` for smart contracts because the owner cannot accidentally transfer the ownership to a mistyped address. Rather than directly transferring to the new owner, the transfer only completes when the new owner accepts ownership.

Remediation

It is recommended to use either `Ownable2Step` or `Ownable2StepUpgradeable` depending on the smart contract.

Bug ID

SSP_4274_54

Bug Type

BLOCK VALUES AS A PROXY FOR TIME

Severity

● Informational

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L107 - L107

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L107 - L107

```
106                 _to,
107                 block.timestamp
108             )[0];
109
```

Description

Contracts often need access to time values to perform certain types of functionality. Values such as `block.timestamp` and `block.number` can be used to determine the current time or the time delta. However, they are not recommended for most use cases.

For `block.number`, as Ethereum block times are generally around 14 seconds, the delta between blocks can be predicted. The block times, on the other hand, do not remain constant and are subject to change for a number of reasons, e.g., fork reorganizations and the difficulty bomb.

Due to variable block times, `block.number` should not be relied on for precise calculations of time.

Remediation

It is recommended to use trusted external time sources, block numbers instead of timestamps, and/or utilizing multiple time sources to increase reliability. These practices can help mitigate risks of timestamp manipulation and inaccurate timing, increasing the reliability and security of the smart contract.

Bug ID	Bug Type	
SSP_4274_55	**BLOCK VALUES AS A PROXY FOR TIME**	
Severity	Action Taken	Detection Method
● Informational	⚠ *Pending Fix*	Automated

Line No.	File Location
L152 - L152	/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L152 - L152

```
151                  address(this),
152                  block.timestamp
153               )[1];
154
```

Description

Contracts often need access to time values to perform certain types of functionality. Values such as `block.timestamp` and `block.number` can be used to determine the current time or the time delta. However, they are not recommended for most use cases.

For `block.number`, as Ethereum block times are generally around 14 seconds, the delta between blocks can be predicted. The block times, on the other hand, do not remain constant and are subject to change for a number of reasons, e.g., fork reorganizations and the difficulty bomb.

Due to variable block times, `block.number` should not be relied on for precise calculations of time.

Remediation

It is recommended to use trusted external time sources, block numbers instead of timestamps, and/or utilizing multiple time sources to increase reliability. These practices can help mitigate risks of timestamp manipulation and inaccurate timing, increasing the reliability and security of the smart contract.

Bug ID	Bug Type	
SSP_4274_55	**BLOCK VALUES AS A PROXY FOR TIME**	
Severity	Action Taken	Detection Method
● Informational	⚠ *Pending Fix*	Automated

Line No.	File Location
L205 - L205	/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L205 - L205

```
204                 address(this),
205                 block.timestamp
206             )[1];
207
```

Description

Contracts often need access to time values to perform certain types of functionality. Values such as `block.timestamp` and `block.number` can be used to determine the current time or the time delta. However, they are not recommended for most use cases.

For `block.number`, as Ethereum block times are generally around 14 seconds, the delta between blocks can be predicted. The block times, on the other hand, do not remain constant and are subject to change for a number of reasons, e.g., fork reorganizations and the difficulty bomb.

Due to variable block times, `block.number` should not be relied on for precise calculations of time.

Remediation

It is recommended to use trusted external time sources, block numbers instead of timestamps, and/or utilizing multiple time sources to increase reliability. These practices can help mitigate risks of timestamp manipulation and inaccurate timing, increasing the reliability and security of the smart contract.

Bug ID	Bug Type	
SSP_4274_56	**BLOCK VALUES AS A PROXY FOR TIME**	
Severity	Action Taken	Detection Method
● Informational	⚠ *Pending Fix*	Automated

Line No.	File Location
L267 - L267	/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L267 - L267

```
266             value: amountToSend
267         }(_amountOutMin, path, _to, block.timestamp);
268         uint amount = IUniswapV2Router02(UNISWAP_V2_ROUTER).getAmountsOut(
269             amountToSend,
```

Description

Contracts often need access to time values to perform certain types of functionality. Values such as `block.timestamp` and `block.number` can be used to determine the current time or the time delta. However, they are not recommended for most use cases.

For `block.number`, as Ethereum block times are generally around 14 seconds, the delta between blocks can be predicted. The block times, on the other hand, do not remain constant and are subject to change for a number of reasons, e.g., fork reorganizations and the difficulty bomb.

Due to variable block times, `block.number` should not be relied on for precise calculations of time.

Remediation

It is recommended to use trusted external time sources, block numbers instead of timestamps, and/or utilizing multiple time sources to increase reliability. These practices can help mitigate risks of timestamp manipulation and inaccurate timing, increasing the reliability and security of the smart contract.

Bug ID	Bug Type	
SSP_4274_57	**BLOCK VALUES AS A PROXY FOR TIME**	
Severity	Action Taken	Detection Method
● Informational	⚠ *Pending Fix*	Automated

Line No.	File Location
L316 - L316	/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L316 - L316

```
315                   address(this),
316                   block.timestamp
317               );
318
```

Description

Contracts often need access to time values to perform certain types of functionality. Values such as `block.timestamp` and `block.number` can be used to determine the current time or the time delta. However, they are not recommended for most use cases.

For `block.number`, as Ethereum block times are generally around 14 seconds, the delta between blocks can be predicted. The block times, on the other hand, do not remain constant and are subject to change for a number of reasons, e.g., fork reorganizations and the difficulty bomb.

Due to variable block times, `block.number` should not be relied on for precise calculations of time.

Remediation

It is recommended to use trusted external time sources, block numbers instead of timestamps, and/or utilizing multiple time sources to increase reliability. These practices can help mitigate risks of timestamp manipulation and inaccurate timing, increasing the reliability and security of the smart contract.

Bug ID

SSP_4274_57

Bug Type

BLOCK VALUES AS A PROXY FOR TIME

Severity

● Informational

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L370 - L370

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L370 - L370

```
369                     address(this),
370                     block.timestamp
371                 );
372
```

Description

Contracts often need access to time values to perform certain types of functionality. Values such as `block.timestamp` and `block.number` can be used to determine the current time or the time delta. However, they are not recommended for most use cases.

For `block.number`, as Ethereum block times are generally around 14 seconds, the delta between blocks can be predicted. The block times, on the other hand, do not remain constant and are subject to change for a number of reasons, e.g., fork reorganizations and the difficulty bomb.

Due to variable block times, `block.number` should not be relied on for precise calculations of time.

Remediation

It is recommended to use trusted external time sources, block numbers instead of timestamps, and/or utilizing multiple time sources to increase reliability. These practices can help mitigate risks of timestamp manipulation and inaccurate timing, increasing the reliability and security of the smart contract.

Bug ID	Bug Type	
SSP_4274_2	**MISSING PAYABLE IN CALL FUNCTION**	
Severity	Action Taken	Detection Method
● Informational	⚠ *Pending Fix*	Automated

Line No.	File Location
L162 - L162	/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L162 - L162

```
161
162         (bool success, ) = maintanierAddress.call{value: maintanierFee}("");
163         require(success, "ETH transfer failed To Maintainer");
164         success = false;
```

Description

The contract is using a `.call()` method to make external calls along with passing some Ether as `msg.value`. Since the function swapWithFeeSell is not marked as `payable`, the transaction might fail if the contract does not have ETH.

Remediation

If the function needs to pass some Ether as `msg.value` inside a function, make sure to set that function as `payable`. No changes are required if the use case is to send Ether from the contract’s balance.

Bug ID

SSP_4274_3

Bug Type

MISSING PAYABLE IN CALL FUNCTION

Severity

● Informational

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L165 - L165

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol

L165 - L165

```
164          success = false;
165          (success, ) = platformAddress.call{value: platformFee}("");
166          require(success, "ETH transfer failed To Platform");
167          success = false;
```

Description

The contract is using a `.call()` method to make external calls along with passing some Ether as `msg.value`. Since the function swapWithFeeSell is not marked as `payable`, the transaction might fail if the contract does not have ETH.

Remediation

If the function needs to pass some Ether as `msg.value` inside a function, make sure to set that function as `payable`. No changes are required if the use case is to send Ether from the contract’s balance.

Bug ID

SSP_4274_4

Bug Type

MISSING PAYABLE IN CALL FUNCTION

Severity

● Informational

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L168 - L168

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L168 - L168

```
167            success = false;
168            (success, ) = _to.call{value: amountToSend}("");
169            require(success, "ETH transfer failed To User");
170
```

Description

The contract is using a `.call()` method to make external calls along with passing some Ether as `msg.value`. Since the function swapWithFeeSell is not marked as `payable`, the transaction might fail if the contract does not have ETH.

Remediation

If the function needs to pass some Ether as `msg.value` inside a function, make sure to set that function as `payable`. No changes are required if the use case is to send Ether from the contract's balance.

Bug ID

SSP_4274_2

Bug Type

MISSING PAYABLE IN CALL FUNCTION

Severity

● Informational

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L215 - L215

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol

L215 - L215

```
214
215            (bool success, ) = maintanierAddress.call{value: maintanierFee}("");
216            require(success, "ETH transfer failed To Maintainer");
217            success = false;
```

Description

The contract is using a `.call()` method to make external calls along with passing some Ether as `msg.value`. Since the function quickSwapWithFeeSell is not marked as `payable`, the transaction might fail if the contract does not have ETH.

Remediation

If the function needs to pass some Ether as `msg.value` inside a function, make sure to set that function as `payable`. No changes are required if the use case is to send Ether from the contract's balance.

Bug ID	Bug Type	
SSP_4274_3	**MISSING PAYABLE IN CALL FUNCTION**	
Severity	Action Taken	Detection Method
● Informational	⚠ *Pending Fix*	Automated

Line No.	File Location
L218 - L218	/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L218 - L218

```
217          success = false;
218          (success, ) = platformAddress.call{value: platformFee}("");
219          require(success, "ETH transfer failed To Platform");
220          success = false;
```

Description

The contract is using a `.call()` method to make external calls along with passing some Ether as `msg.value`. Since the function quickSwapWithFeeSell is not marked as `payable`, the transaction might fail if the contract does not have ETH.

Remediation

If the function needs to pass some Ether as `msg.value` inside a function, make sure to set that function as `payable`. No changes are required if the use case is to send Ether from the contract's balance.

Bug ID	Bug Type	
SSP_4274_4	**MISSING PAYABLE IN CALL FUNCTION**	
Severity	Action Taken	Detection Method
● Informational	⚠ *Pending Fix*	Automated

Line No.	File Location
L221 - L221	/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L221 - L221

```
220         success = false;
221         (success, ) = _to.call{value: amountToSend}("");
222         require(success, "ETH transfer failed To User");
223
```

Description

The contract is using a `.call()` method to make external calls along with passing some Ether as `msg.value`. Since the function quickSwapWithFeeSell is not marked as `payable`, the transaction might fail if the contract does not have ETH.

Remediation

If the function needs to pass some Ether as `msg.value` inside a function, make sure to set that function as `payable`. No changes are required if the use case is to send Ether from the contract’s balance.

Bug ID	Bug Type	
SSP_4274_2	MISSING PAYABLE IN CALL FUNCTION	

Severity	Action Taken	Detection Method
● Informational	⚠ *Pending Fix*	Automated

Line No.	File Location
L326 - L326	/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L326 - L326

```
325
326         (bool success, ) = maintanierAddress.call{value: maintanierFee}("");
327         require(success, "ETH transfer failed To Maintainer");
328         success = false;
```

Description

The contract is using a `.call()` method to make external calls along with passing some Ether as `msg.value`. Since the function swapWithSellTaxToken is not marked as `payable`, the transaction might fail if the contract does not have ETH.

Remediation

If the function needs to pass some Ether as `msg.value` inside a function, make sure to set that function as `payable`. No changes are required if the use case is to send Ether from the contract's balance.

Bug ID	Bug Type	
SSP_4274_3	**MISSING PAYABLE IN CALL FUNCTION**	
Severity	Action Taken	Detection Method
● Informational	⚠ *Pending Fix*	Automated

Line No.	File Location
L329 - L329	/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L329 - L329

```
328            success = false;
329            (success, ) = platformAddress.call{value: platformFee}("");
330            require(success, "ETH transfer failed To Platform");
331           success = false;
```

Description

The contract is using a `.call()` method to make external calls along with passing some Ether as `msg.value`. Since the function swapWithSellTaxToken is not marked as `payable`, the transaction might fail if the contract does not have ETH.

Remediation

If the function needs to pass some Ether as `msg.value` inside a function, make sure to set that function as `payable`. No changes are required if the use case is to send Ether from the contract's balance.

Bug ID

SSP_4274_4

Bug Type

MISSING PAYABLE IN CALL FUNCTION

Severity

● Informational

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L332 - L332

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L332 - L332

```
        success = false;
        (success, ) = _to.call{value: amountToSend}("");
        require(success, "ETH transfer failed To User");

```

Description

The contract is using a `.call()` method to make external calls along with passing some Ether as `msg.value`. Since the function swapWithSellTaxToken is not marked as `payable`, the transaction might fail if the contract does not have ETH.

Remediation

If the function needs to pass some Ether as `msg.value` inside a function, make sure to set that function as `payable`. No changes are required if the use case is to send Ether from the contract's balance.

Bug ID	Bug Type	
SSP_4274_2	**MISSING PAYABLE IN CALL FUNCTION**	
Severity	Action Taken	Detection Method
● Informational	⚠ *Pending Fix*	Automated

Line No.	File Location
L380 - L380	/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L380 - L380

```
379
380         (bool success, ) = maintanierAddress.call{value: maintanierFee}("");
381         require(success, "ETH transfer failed To Maintainer");
382         success = false;
```

Description

The contract is using a `.call()` method to make external calls along with passing some Ether as `msg.value`. Since the function quickSwapWithSellTaxToken is not marked as `payable`, the transaction might fail if the contract does not have ETH.

Remediation

If the function needs to pass some Ether as `msg.value` inside a function, make sure to set that function as `payable`. No changes are required if the use case is to send Ether from the contract's balance.

Bug ID

SSP_4274_3

Bug Type

MISSING PAYABLE IN CALL FUNCTION

Severity

● Informational

Action Taken

Pending Fix

Detection Method

Automated

Line No.

L383 - L383

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L383 - L383

```
382         success = false;
383         (success, ) = platformAddress.call{value: platformFee}("");
384         require(success, "ETH transfer failed To Platform");
385         success = false;
```

Description

The contract is using a `.call()` method to make external calls along with passing some Ether as `msg.value`. Since the function quickSwapWithSellTaxToken is not marked as `payable`, the transaction might fail if the contract does not have ETH.

Remediation

If the function needs to pass some Ether as `msg.value` inside a function, make sure to set that function as `payable`. No changes are required if the use case is to send Ether from the contract’s balance.

Bug ID	Bug Type	
SSP_4274_4	**MISSING PAYABLE IN CALL FUNCTION**	
Severity	Action Taken	Detection Method
● Informational	⚠ *Pending Fix*	Automated

Line No.	File Location
L386 - L386	/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L386 - L386

```
385         success = false;
386         (success, ) = _to.call{value: amountToSend}("");
387         require(success, "ETH transfer failed To User");
388
```

Description

The contract is using a `.call()` method to make external calls along with passing some Ether as `msg.value`. Since the function quickSwapWithSellTaxToken is not marked as `payable`, the transaction might fail if the contract does not have ETH.

Remediation

If the function needs to pass some Ether as `msg.value` inside a function, make sure to set that function as `payable`. No changes are required if the use case is to send Ether from the contract's balance.

Bug ID	Bug Type	
SSP_4274_6	MISSING UNDERSCORE IN NAMING VARIABLES	

Severity	Action Taken	Detection Method
● Informational	⚠ *Pending Fix*	Automated

Line No.	File Location
L18 - L18	/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L18 - L18

```
17        //Address of the fund receiver
18        address private platformAddress;
19
20        address private maintanierAddress;
```

Description

Solidity style guide suggests using underscores as the prefix for non-external functions and state variables (private or internal) but the contract was not found to be following the same.

Remediation

It is recommended to use an underscore for internal and private variables and functions to be in accordance with the Solidity style guide which will also make the code much easier to read.

Bug ID

SSP_4274_7

Bug Type

MISSING UNDERSCORE IN NAMING VARIABLES

Severity

● Informational

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L20 - L20

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L20 - L20

```
19
20        address private maintanierAddress;
21
22        constructor() {
```

Description

Solidity style guide suggests using underscores as the prefix for non-external functions and state variables (private or internal) but the contract was not found to be following the same.

Remediation

It is recommended to use an underscore for internal and private variables and functions to be in accordance with the Solidity style guide which will also make the code much easier to read.

Bug ID

SSP_4274_8

Bug Type

MISSING UNDERSCORE IN NAMING VARIABLES

Severity

● Informational

Action Taken

Pending Fix

Detection Method

Automated

Line No.

L407 - L423

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol

L407 - L423

```
406
407         function percentageCalculation(
408             uint256 _amountIn
409         )
410             internal
411             pure
412             returns (
413                 uint256 deductionAmount,
414                 uint256 maintanierFee,
415                 uint256 platformFee,
416                 uint256 amountToSwap
417             )
418         {
419             deductionAmount = (_amountIn * 200) / 10000; // 2% deduction
420             maintanierFee = (deductionAmount * 50) / 100;
421             platformFee = deductionAmount - maintanierFee;
422             amountToSwap = _amountIn - deductionAmount;
423         }
424
425         /**
```

Description

Solidity style guide suggests using underscores as the prefix for non-external functions and state variables (private or i nternal) but the contract was not found to be following the same.

Remediation

It is recommended to use an underscore for internal and private variables and functions to be in accordance with the So lidity style guide which will also make the code much easier to read.

Bug ID	Bug Type	
SSP_4274_36	**UNUSED RECEIVE FALLBACK**	
Severity	Action Track	Detection Method

Severity	Action Taken	Detection Method
● Informational	⚠ *Pending Fix*	Automated

Line No.	File Location
L466 - L468	/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L466 - L468

```
465
466        receive() external payable {
467            // React to receiving ether
468        }
469
470        function withdrawEther(
```

Description

The contract was found to be defining an empty receive function.
It is not recommended to leave them empty unless there's a specific use case such as to receive Ether via an empty `receive()` function.

Remediation

It is recommended to go through the code to make sure these functions are properly implemented and are not missing any validations in the definition.

Bug ID

SSP_4274_27

Bug Type

USE CALL INSTEAD OF TRANSFER OR SEND

Severity

● Informational

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L480 - L480

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L480 - L480

```
479
480             recipient.transfer(amount);
481         }
482     }
```

Description

The contract was found to be using `transfer` or `send` function call. This is unsafe as `transfer` has hard coded g as budget and can fail if the user is a smart contract.

Remediation

It is recommended to use `call` which does not have any hardcoded gas.

Bug ID	Bug Type	
SSP_4274_26	**CHEAPER CONDITIONAL OPERATORS**	
Severity	Action Taken	Detection Method
● Gas	⚠ *Pending Fix*	Automated

Line No.	File Location
L83 - L83	/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L83 - L83

```
82      ) external payable ZeroAddress(_to) ZeroAmount(_amountOutMin) {
83          require(msg.value > 0, "BC:Invalid ETH Amount");
84          // Construct the token swap path
85          address[] memory path;
```

Description

During compilation, `x != 0` is cheaper than `x > 0` for unsigned integers in solidity inside conditional statements.

Remediation

Consider using `x != 0` in place of `x > 0` in `uint` wherever possible.

Bug ID

SSP_4274_26

Bug Type

CHEAPER CONDITIONAL OPERATORS

Severity

● Gas

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L245 - L245

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L245 - L245

```
244      ) external payable ZeroAddress(_to) ZeroAmount(_amountOutMin) {
245          require(msg.value > 0, "BC:Invalid ETH Amount");
246          // Construct the token swap path
247          address[] memory path;
```

Description

During compilation, `x != 0` is cheaper than `x > 0` for unsigned integers in solidity inside conditional statements.

Remediation

Consider using `x != 0` in place of `x > 0` in `uint` wherever possible.

Bug ID	Bug Type
SSP_4274_24	**CHEAPER INEQUALITIES IN REQUIRE()**

Severity	Action Take	Detection Method
● Gas	⚠ *Pending Fix*	Automated

Line No.	File Location
L476 - L476	/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L476 - L476

```
475         require(
476             address(this).balance >= amount,
477             "Insufficient balance in the contract"
478         );
```

Description

The contract was found to be performing comparisons using inequalities inside the `require` statement. When inside t he `require` statements, non-strict inequalities `(>=, <=)` are usually costlier than strict equalities `(>, <)`.

Remediation

It is recommended to go through the code logic, and, if possible, modify the non-strict inequalities with the strict ones to save `~3` gas as long as the logic of the code is not affected.

Bug ID	Bug Type	
SSP_4274_1	**DEFINE CONSTRUCTOR AS PAYABLE**	
Severity	Action Taken	Detection Method
● Gas	⚠ *Pending Fix*	Automated

Line No.	File Location
L22 - L24	/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L22 - L24

```
21
22        constructor() {
23            _disableInitializers();
24        }
25
26        modifier ZeroAddress(address _account) {
```

Description

Developers can save around 10 opcodes and some gas if the constructors are defined as payable.
However, it should be noted that it comes with risks because payable constructors can accept ETH during deployment.

Remediation

It is suggested to mark the constructors as payable to save some gas. Make sure it does not lead to any adverse effects in case an upgrade pattern is involved.

Bug ID

SSP_4274_22

Bug Type

FUNCTION SHOULD RETURN STRUCT

Severity

● Gas

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L407 - L423

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L407 - L423

```
406
407        function percentageCalculation(
408            uint256 _amountIn
409        )
410            internal
411            pure
412            returns (
413                uint256 deductionAmount,
414                uint256 maintanierFee,
415                uint256 platformFee,
416                uint256 amountToSwap
417            )
418        {
419            deductionAmount = (_amountIn * 200) / 10000; // 2% deduction
420            maintanierFee = (deductionAmount * 50) / 100;
421            platformFee = deductionAmount - maintanierFee;
422            amountToSwap = _amountIn - deductionAmount;
423        }
424
425        /**
```

Description

The function percentageCalculation was detected to be returning multiple values.
Consider using a `struct` instead of multiple return values for the function. It can improve code readability.

Remediation

Use `struct` for returning multiple values inside a function, which returns several parameters and improves code read ability.

Bug ID

SSP_4274_39

Bug Type

LONG REQUIRE/REVERT STRINGS

Severity	Action Taken	Detection Method
● Gas	⚠ *Pending Fix*	Automated

Line No.	File Location
L98 - L98	/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L98 - L98

```
97            (bool success, ) = maintanierAddress.call{value: maintanierFee}("");
98            require(success, "ETH transfer failed To Maintainer");
99            (success, ) = platformAddress.call{value: platformFee}("");
100            require(success, "ETH transfer failed To Maintainer");
```

Description

The `require()` and `revert()` functions take an input string to show errors if the validation fails.
This strings inside these functions that are longer than `32 bytes` require at least one additional `MSTORE`, along with additional overhead for computing memory offset, and other parameters.

Remediation

It is recommended to short the strings passed inside `require()` and `revert()` to fit under `32 bytes`. This will decrease the gas usage at the time of deployment and at runtime when the validation condition is met.

Bug ID

SSP_4274_40

Bug Type

LONG REQUIRE/REVERT STRINGS

Severity

● Gas

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L100 - L100

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L100 - L100

```
99          (success, ) = platformAddress.call{value: platformFee}("");
100          require(success, "ETH transfer failed To Maintainer");
101
102          uint256 _amountOut = IUniswapV2Router02(UNISWAP_V2_ROUTER)
```

Description

The `require()` and `revert()` functions take an input string to show errors if the validation fails.
This strings inside these functions that are longer than `32 bytes` require at least one additional `MSTORE`, along with additional overhead for computing memory offset, and other parameters.

Remediation

It is recommended to short the strings passed inside `require()` and `revert()` to fit under `32 bytes`. This will decrease the gas usage at the time of deployment and at runtime when the validation condition is met.

Bug ID

SSP_4274_41

Bug Type

LONG REQUIRE/REVERT STRINGS

Severity

● Gas

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L163 - L163

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L163 - L163

```
162         (bool success, ) = maintanierAddress.call{value: maintanierFee}("");
163         require(success, "ETH transfer failed To Maintainer");
164         success = false;
165         (success, ) = platformAddress.call{value: platformFee}("");
```

Description

The `require()` and `revert()` functions take an input string to show errors if the validation fails.
This strings inside these functions that are longer than `32 bytes` require at least one additional `MSTORE`, along with additional overhead for computing memory offset, and other parameters.

Remediation

It is recommended to short the strings passed inside `require()` and `revert()` to fit under `32 bytes`. This will dec rease the gas usage at the time of deployment and at runtime when the validation condition is met.

Bug ID

SSP_4274_41

Bug Type

LONG REQUIRE/REVERT STRINGS

Severity

● Gas

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L216 - L216

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L216 - L216

```
215         (bool success, ) = maintanierAddress.call{value: maintanierFee}("");
216         require(success, "ETH transfer failed To Maintainer");
217         success = false;
218         (success, ) = platformAddress.call{value: platformFee}("");
```

Description

The `require()` and `revert()` functions take an input string to show errors if the validation fails.
This strings inside these functions that are longer than `32 bytes` require at least one additional `MSTORE`, along with additional overhead for computing memory offset, and other parameters.

Remediation

It is recommended to short the strings passed inside `require()` and `revert()` to fit under `32 bytes`. This will decrease the gas usage at the time of deployment and at runtime when the validation condition is met.

Bug ID	Bug Type	
SSP_4274_39	LONG REQUIRE/REVERT STRINGS	

Severity	Action Taken	Detection Method
● Gas	⚠ *Pending Fix*	Automated

Line No.	File Location
L260 - L260	/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L260 - L260

```
259           (bool success, ) = maintanierAddress.call{value: maintanierFee}("");
260           require(success, "ETH transfer failed To Maintainer");
261           (success, ) = platformAddress.call{value: platformFee}("");
262           require(success, "ETH transfer failed To Maintainer");
```

Description

The `require()` and `revert()` functions take an input string to show errors if the validation fails.
This strings inside these functions that are longer than `32 bytes` require at least one additional `MSTORE`, along with additional overhead for computing memory offset, and other parameters.

Remediation

It is recommended to short the strings passed inside `require()` and `revert()` to fit under `32 bytes`. This will dec rease the gas usage at the time of deployment and at runtime when the validation condition is met.

Bug ID

SSP_4274_40

Bug Type

LONG REQUIRE/REVERT STRINGS

Severity

● Gas

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L262 - L262

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L262 - L262

```
261         (success, ) = platformAddress.call{value: platformFee}("");
262         require(success, "ETH transfer failed To Maintainer");
263
264         IUniswapV2Router02(UNISWAP_V2_ROUTER)
```

Description

The `require()` and `revert()` functions take an input string to show errors if the validation fails.
This strings inside these functions that are longer than `32 bytes` require at least one additional `MSTORE`, along with additional overhead for computing memory offset, and other parameters.

Remediation

It is recommended to short the strings passed inside `require()` and `revert()` to fit under `32 bytes`. This will decrease the gas usage at the time of deployment and at runtime when the validation condition is met.

Bug ID	Bug Type	
SSP_4274_41	**LONG REQUIRE/REVERT STRINGS**	
Severity	Action Taken	Detection Method
● Gas	⚠ *Pending Fix*	Automated
Line No.	File Location	
L327 - L327	/BuyContract (1).sol	

Affected Code

/BuyContract (1).sol L327 - L327

```
326         (bool success, ) = maintanierAddress.call{value: maintanierFee}("");
327         require(success, "ETH transfer failed To Maintainer");
328         success = false;
329         (success, ) = platformAddress.call{value: platformFee}("");
```

Description

The `require()` and `revert()` functions take an input string to show errors if the validation fails.
This strings inside these functions that are longer than `32 bytes` require at least one additional `MSTORE`, along with additional overhead for computing memory offset, and other parameters.

Remediation

It is recommended to short the strings passed inside `require()` and `revert()` to fit under `32 bytes`. This will dec rease the gas usage at the time of deployment and at runtime when the validation condition is met.

Bug ID	Bug Type	
SSP_4274_41	**LONG REQUIRE/REVERT STRINGS**	
Severity	Action Taken	Detection Method
● Gas	⚠ *Pending Fix*	Automated

Line No.	File Location
L381 - L381	/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L381 - L381

```
380         (bool success, ) = maintanierAddress.call{value: maintanierFee}("");
381         require(success, "ETH transfer failed To Maintainer");
382         success = false;
383         (success, ) = platformAddress.call{value: platformFee}("");
```

Description

The `require()` and `revert()` functions take an input string to show errors if the validation fails.
This strings inside these functions that are longer than `32 bytes` require at least one additional `MSTORE`, along with additional overhead for computing memory offset, and other parameters.

Remediation

It is recommended to short the strings passed inside `require()` and `revert()` to fit under `32 bytes`. This will dec rease the gas usage at the time of deployment and at runtime when the validation condition is met.

Bug ID	Bug Type	
SSP_4274_53	**LONG REQUIRE/REVERT STRINGS**	
Severity	Action Taken	Detection Method
● Gas	⚠ *Pending Fix*	Automated

Line No.	File Location
L475 - L478	/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L475 - L478

```
474          require(recipient != address(0), "Invalid recipient address");
475          require(
476              address(this).balance >= amount,
477              "Insufficient balance in the contract"
478          );
479
480          recipient.transfer(amount);
```

Description

The `require()` and `revert()` functions take an input string to show errors if the validation fails.
This strings inside these functions that are longer than `32 bytes` require at least one additional `MSTORE`, along with additional overhead for computing memory offset, and other parameters.

Remediation

It is recommended to short the strings passed inside `require()` and `revert()` to fit under `32 bytes`. This will dec rease the gas usage at the time of deployment and at runtime when the validation condition is met.

Bug ID

SSP_4274_10

Bug Type

STORAGE VARIABLE CACHING IN MEMORY

Severity

● Gas

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L78 - L117

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L78 - L117

```
    function swapWithFeeBuy(
        address _tokenOut,
        uint256 _amountOutMin,
        address _to
    ) external payable ZeroAddress(_to) ZeroAmount(_amountOutMin) {
        require(msg.value > 0, "BC:Invalid ETH Amount");
        // Construct the token swap path
        address[] memory path;

        path = new address[](2);
        path[0] = WETH;
        path[1] = _tokenOut;

        (
            ,
            uint256 maintanierFee,
            uint256 platformFee,
            uint256 amountToSend
        ) = percentageCalculation(msg.value);
        (bool success, ) = maintanierAddress.call{value: maintanierFee}("");
        require(success, "ETH transfer failed To Maintainer");
        (success, ) = platformAddress.call{value: platformFee}("");
         require(success, "ETH transfer failed To Maintainer");

         uint256 _amountOut = IUniswapV2Router02(UNISWAP_V2_ROUTER)
             .swapExactETHForTokens{value: amountToSend}(
             _amountOutMin,
             path,
```

/BuyContract (1).sol L78 - L117

```
            .swapExactETHForTokens{value: amountToSend}(
            _amountOutMin,
            path,
            _to,
            block.timestamp
        )[0];

        emit TokensSwapped(
            WETH, // ETH address
            _tokenOut,
            amountToSend,
            _amountOut,
            _to
        );
    }

    /**
```

Description

The contract `BuyContract` is using the state variable `WETH` multiple times in the function `swapWithFeeBuy`. `SLOADs` are expensive (100 gas after the 1st one) compared to `MLOAD` / `MSTORE` (3 gas each).

Remediation

Storage variables read multiple times inside a function should instead be cached in the memory the first time (costing 1 `SLOAD`) and then read from this cache to avoid multiple `SLOADs`.

Bug ID	Bug Type	
SSP_4274_11	STORAGE VARIABLE CACHING IN MEMORY	
Severity	**Action Taken**	**Detection Method**
● Gas	⚠ *Pending Fix*	Automated

Line No.	File Location
L127 - L178	/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L127 - L178

```

    function swapWithFeeSell(
        address _tokenIn,
        uint256 _amountIn,
        uint256 _amountOutMin,
        address _to
    )
        external
        ZeroAddress(_to)
        ZeroAmount(_amountIn)
        ZeroAmount(_amountOutMin)
    {
        // Construct the token swap path
        address[] memory path = new address[](2);
        path[0] = _tokenIn;
        path[1] = WETH;

        IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
        IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);

        uint256 amount = IUniswapV2Router02(UNISWAP_V2_ROUTER)
            .swapExactTokensForETH(
                _amountIn,
                _amountOutMin,
                path,
                address(this),
                block.timestamp
            )[1];

```

/BuyContract (1).sol L127 - L178

```
                block.timestamp
            )[1];

        (
            ,
            uint256 maintanierFee,
            uint256 platformFee,
            uint256 amountToSend
        ) = percentageCalculation(amount);

        (bool success, ) = maintanierAddress.call{value: maintanierFee}("");
        require(success, "ETH transfer failed To Maintainer");
        success = false;
        (success, ) = platformAddress.call{value: platformFee}("");
        require(success, "ETH transfer failed To Platform");
        success = false;
        (success, ) = _to.call{value: amountToSend}("");
        require(success, "ETH transfer failed To User");

        emit TokensSwapped(
            _tokenIn,
            WETH, // ETH address
            _amountIn,
            amountToSend,
            _to
        );
    }

    /**
```

Description

The contract `BuyContract` is using the state variable `UNISWAP_V2_ROUTER` multiple times in the function `swapWithFeeSell`.
`SLOADs` are expensive (100 gas after the 1st one) compared to `MLOAD` / `MSTORE` (3 gas each).

Remediation

Storage variables read multiple times inside a function should instead be cached in the memory the first time (costing 1 `SLOAD`) and then read from this cache to avoid multiple `SLOADs`.

Bug ID	Bug Type	
SSP_4274_11	STORAGE VARIABLE CACHING IN MEMORY	
Severity	Action Taken	Detection Method
● Gas	⚠ *Pending Fix*	Automated

Line No.	File Location
L127 - L178	/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L127 - L178

```

    function swapWithFeeSell(
        address _tokenIn,
        uint256 _amountIn,
        uint256 _amountOutMin,
        address _to
    )
        external
        ZeroAddress(_to)
        ZeroAmount(_amountIn)
        ZeroAmount(_amountOutMin)
    {
        // Construct the token swap path
        address[] memory path = new address[](2);
        path[0] = _tokenIn;
        path[1] = WETH;

        IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
        IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);

        uint256 amount = IUniswapV2Router02(UNISWAP_V2_ROUTER)
            .swapExactTokensForETH(
                _amountIn,
                _amountOutMin,
                path,
                address(this),
                block.timestamp
            )[1];

```

/BuyContract (1).sol L127 - L178

```
                block.timestamp
            )[1];

        (
            ,
            uint256 maintanierFee,
            uint256 platformFee,
            uint256 amountToSend
        ) = percentageCalculation(amount);

        (bool success, ) = maintanierAddress.call{value: maintanierFee}("");
        require(success, "ETH transfer failed To Maintainer");
        success = false;
        (success, ) = platformAddress.call{value: platformFee}("");
        require(success, "ETH transfer failed To Platform");
        success = false;
        (success, ) = _to.call{value: amountToSend}("");
        require(success, "ETH transfer failed To User");

        emit TokensSwapped(
            _tokenIn,
            WETH, // ETH address
            _amountIn,
            amountToSend,
            _to
        );
    }

    /**
```

Description

The contract `BuyContract` is using the state variable `WETH` multiple times in the function `swapWithFeeSell`. `SLOADs` are expensive (100 gas after the 1st one) compared to `MLOAD` / `MSTORE` (3 gas each).

Remediation

Storage variables read multiple times inside a function should instead be cached in the memory the first time (costing 1 `SLOAD`) and then read from this cache to avoid multiple `SLOADs`.

Bug ID

SSP_4274_21

Bug Type

STORAGE VARIABLE CACHING IN MEMORY

Severity

● Gas

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L186 - L231

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L186 - L231

```

    function quickSwapWithFeeSell(
        address _tokenIn,
        address _to
    ) external ZeroAddress(_to) {
        // Construct the token swap path
        uint256 _amountIn = IERC20(_tokenIn).balanceOf(msg.sender);
        address[] memory path = new address[](2);
        path[0] = _tokenIn;
        path[1] = WETH;

        IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
        IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);

        uint256 amount = IUniswapV2Router02(UNISWAP_V2_ROUTER)
            .swapExactTokensForETH(
                _amountIn,
                0,
                path,
                address(this),
                block.timestamp
            )[1];

        (
            ,
            uint256 maintanierFee,
            uint256 platformFee,
            uint256 amountToSend
        ) = percentageCalculation(amount);
```

/BuyContract (1).sol L186 - L231

```
            uint256 platformFee,
            uint256 amountToSend
        ) = percentageCalculation(amount);

        (bool success, ) = maintanierAddress.call{value: maintanierFee}("");
        require(success, "ETH transfer failed To Maintainer");
        success = false;
        (success, ) = platformAddress.call{value: platformFee}("");
        require(success, "ETH transfer failed To Platform");
        success = false;
        (success, ) = _to.call{value: amountToSend}("");
        require(success, "ETH transfer failed To User");

        emit TokensSwapped(
            _tokenIn,
            WETH, // ETH address
            _amountIn,
            amountToSend,
            _to
        );
    }

    /**
```

Description

The contract `BuyContract` is using the state variable `UNISWAP_V2_ROUTER` multiple times in the function `quickSwapWithFeeSell`.
`SLOADs` are expensive (100 gas after the 1st one) compared to `MLOAD` / `MSTORE` (3 gas each).

Remediation

Storage variables read multiple times inside a function should instead be cached in the memory the first time (costing 1 `SLOAD`) and then read from this cache to avoid multiple `SLOADs`.

Bug ID

SSP_4274_21

Bug Type

STORAGE VARIABLE CACHING IN MEMORY

Severity

● Gas

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L186 - L231

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L186 - L231

```

    function quickSwapWithFeeSell(
        address _tokenIn,
        address _to
    ) external ZeroAddress(_to) {
        // Construct the token swap path
        uint256 _amountIn = IERC20(_tokenIn).balanceOf(msg.sender);
        address[] memory path = new address[](2);
        path[0] = _tokenIn;
        path[1] = WETH;

        IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
        IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);

        uint256 amount = IUniswapV2Router02(UNISWAP_V2_ROUTER)
            .swapExactTokensForETH(
                _amountIn,
                0,
                path,
                address(this),
                block.timestamp
            )[1];

        (
            ,
            uint256 maintanierFee,
            uint256 platformFee,
            uint256 amountToSend
        ) = percentageCalculation(amount);
```

/BuyContract (1).sol L186 - L231

```
            uint256 platformFee,
            uint256 amountToSend
        ) = percentageCalculation(amount);

        (bool success, ) = maintanierAddress.call{value: maintanierFee}("");
        require(success, "ETH transfer failed To Maintainer");
        success = false;
        (success, ) = platformAddress.call{value: platformFee}("");
        require(success, "ETH transfer failed To Platform");
        success = false;
        (success, ) = _to.call{value: amountToSend}("");
        require(success, "ETH transfer failed To User");

        emit TokensSwapped(
            _tokenIn,
            WETH, // ETH address
            _amountIn,
            amountToSend,
            _to
        );
    }

    /**
```

Description

The contract `BuyContract` is using the state variable `WETH` multiple times in the function `quickSwapWithFeeSell`.

`SLOADs` are expensive (100 gas after the 1st one) compared to `MLOAD` / `MSTORE` (3 gas each).

Remediation

Storage variables read multiple times inside a function should instead be cached in the memory the first time (costing 1 `SLOAD`) and then read from this cache to avoid multiple `SLOADs`.

Bug ID

SSP_4274_28

Bug Type

STORAGE VARIABLE CACHING IN MEMORY

Severity

● Gas

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L240 - L280

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L240 - L280

```
function swapWithBuyTaxToken(
    address _tokenOut,
    uint256 _amountOutMin,
    address _to
) external payable ZeroAddress(_to) ZeroAmount(_amountOutMin) {
    require(msg.value > 0, "BC:Invalid ETH Amount");
    // Construct the token swap path
    address[] memory path;

    path = new address[](2);
    path[0] = WETH;
    path[1] = _tokenOut;

    (
        ,
        uint256 maintanierFee,
        uint256 platformFee,
        uint256 amountToSend
    ) = percentageCalculation(msg.value);
    (bool success, ) = maintanierAddress.call{value: maintanierFee}("");
    require(success, "ETH transfer failed To Maintainer");
    (success, ) = platformAddress.call{value: platformFee}("");
    require(success, "ETH transfer failed To Maintainer");

    IUniswapV2Router02(UNISWAP_V2_ROUTER)
        .swapExactETHForTokensSupportingFeeOnTransferTokens{
        value: amountToSend
    }(_amountOutMin, path, _to, block.timestamp);
```

/BuyContract (1).sol L240 - L280

```
            .swapExactETHForTokensSupportingFeeOnTransferTokens{
            value: amountToSend
        }(_amountOutMin, path, _to, block.timestamp);
        uint amount = IUniswapV2Router02(UNISWAP_V2_ROUTER).getAmountsOut(
            amountToSend,
            path
        )[0];

        emit TokensSwapped(
            WETH, // ETH address
            _tokenOut,
            amountToSend,
            amount,
            _to
        );
    }

    /**
```

Description

The contract `BuyContract` is using the state variable `UNISWAP_V2_ROUTER` multiple times in the function `swapWithBuyTaxToken`.
`SLOADs` are expensive (100 gas after the 1st one) compared to `MLOAD` / `MSTORE` (3 gas each).

Remediation

Storage variables read multiple times inside a function should instead be cached in the memory the first time (costing 1 `SLOAD`) and then read from this cache to avoid multiple `SLOADs`.

Bug ID

SSP_4274_28

Bug Type

STORAGE VARIABLE CACHING IN MEMORY

Severity

● Gas

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L240 - L280

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol

L240 - L280

```

    function swapWithBuyTaxToken(
        address _tokenOut,
        uint256 _amountOutMin,
        address _to
    ) external payable ZeroAddress(_to) ZeroAmount(_amountOutMin) {
        require(msg.value > 0, "BC:Invalid ETH Amount");
        // Construct the token swap path
        address[] memory path;

        path = new address[](2);
        path[0] = WETH;
        path[1] = _tokenOut;

        (
            ,
            uint256 maintanierFee,
            uint256 platformFee,
            uint256 amountToSend
        ) = percentageCalculation(msg.value);
        (bool success, ) = maintanierAddress.call{value: maintanierFee}("");
        require(success, "ETH transfer failed To Maintainer");
        (success, ) = platformAddress.call{value: platformFee}("");
        require(success, "ETH transfer failed To Maintainer");

        IUniswapV2Router02(UNISWAP_V2_ROUTER)
            .swapExactETHForTokensSupportingFeeOnTransferTokens{
            value: amountToSend
        }(_amountOutMin, path, _to, block.timestamp);
```

/BuyContract (1).sol L240 - L280

```
            .swapExactETHForTokensSupportingFeeOnTransferTokens{
            value: amountToSend
        }(_amountOutMin, path, _to, block.timestamp);
        uint amount = IUniswapV2Router02(UNISWAP_V2_ROUTER).getAmountsOut(
            amountToSend,
            path
        )[0];

        emit TokensSwapped(
            WETH, // ETH address
            _tokenOut,
            amountToSend,
            amount,
            _to
        );
    }

    /**
```

Description

The contract `BuyContract` is using the state variable `WETH` multiple times in the function `swapWithBuyTaxToken`. `SLOADs` are expensive (100 gas after the 1st one) compared to `MLOAD` / `MSTORE` (3 gas each).

Remediation

Storage variables read multiple times inside a function should instead be cached in the memory the first time (costing 1 `SLOAD`) and then read from this cache to avoid multiple `SLOADs`.

Bug ID

SSP_4274_29

Bug Type

STORAGE VARIABLE CACHING IN MEMORY

Severity

● Gas

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L290 - L342

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L290 - L342

```

function swapWithSellTaxToken(
    address _tokenIn,
    uint256 _amountIn,
    uint256 _amountOutMin,
    address _to,
    uint _afterTax
)
    external
    ZeroAddress(_to)
    ZeroAmount(_amountIn)
    ZeroAmount(_amountOutMin)
{
    // Construct the token swap path
    address[] memory path = new address[](2);
    path[0] = _tokenIn;
    path[1] = WETH;

    IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
    IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);

    IUniswapV2Router02(UNISWAP_V2_ROUTER)
        .swapExactTokensForETHSupportingFeeOnTransferTokens(
            IERC20(_tokenIn).balanceOf(address(this)),
            _amountOutMin,
            path,
            address(this),
            block.timestamp
        );
```

/BuyContract (1).sol L290 - L342

```
                address(this),
                block.timestamp
            );

        (
            ,
            uint256 maintanierFee,
            uint256 platformFee,
            uint256 amountToSend
        ) = percentageCalculation(_afterTax);

        (bool success, ) = maintanierAddress.call{value: maintanierFee}("");
        require(success, "ETH transfer failed To Maintainer");
        success = false;
        (success, ) = platformAddress.call{value: platformFee}("");
        require(success, "ETH transfer failed To Platform");
        success = false;
        (success, ) = _to.call{value: amountToSend}("");
        require(success, "ETH transfer failed To User");

        emit TokensSwapped(
            _tokenIn,
            WETH, // ETH address
            _amountIn,
            amountToSend,
            _to
        );
    }

    /**
```

Description

The contract `BuyContract` is using the state variable `UNISWAP_V2_ROUTER` multiple times in the function `swapWithSellTaxToken`.
`SLOADs` are expensive (100 gas after the 1st one) compared to `MLOAD` / `MSTORE` (3 gas each).

Remediation

Storage variables read multiple times inside a function should instead be cached in the memory the first time (costing 1 `SLOAD`) and then read from this cache to avoid multiple `SLOADs`.

Bug ID

SSP_4274_29

Bug Type

STORAGE VARIABLE CACHING IN MEMORY

Severity

● Gas

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L290 - L342

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L290 - L342

```
289
290         function swapWithSellTaxToken(
291             address _tokenIn,
292             uint256 _amountIn,
293             uint256 _amountOutMin,
294             address _to,
295             uint _afterTax
296         )
297             external
298             ZeroAddress(_to)
299             ZeroAmount(_amountIn)
300             ZeroAmount(_amountOutMin)
301         {
302             // Construct the token swap path
303             address[] memory path = new address[](2);
304             path[0] = _tokenIn;
305             path[1] = WETH;
306
307             IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
308             IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);
309
310             IUniswapV2Router02(UNISWAP_V2_ROUTER)
311                 .swapExactTokensForETHSupportingFeeOnTransferTokens(
312                     IERC20(_tokenIn).balanceOf(address(this)),
313                     _amountOutMin,
314                     path,
315                     address(this),
316                     block.timestamp
317                 );
```

/BuyContract (1).sol L290 - L342

```
                address(this),
                block.timestamp
            );

        (
            ,
            uint256 maintanierFee,
            uint256 platformFee,
            uint256 amountToSend
        ) = percentageCalculation(_afterTax);

        (bool success, ) = maintanierAddress.call{value: maintanierFee}("");
        require(success, "ETH transfer failed To Maintainer");
        success = false;
        (success, ) = platformAddress.call{value: platformFee}("");
        require(success, "ETH transfer failed To Platform");
        success = false;
        (success, ) = _to.call{value: amountToSend}("");
        require(success, "ETH transfer failed To User");

        emit TokensSwapped(
            _tokenIn,
            WETH, // ETH address
            _amountIn,
            amountToSend,
            _to
        );
    }

    /**
```

Description

The contract `BuyContract` is using the state variable `WETH` multiple times in the function `swapWithSellTaxToken`.

`SLOADs` are expensive (100 gas after the 1st one) compared to `MLOAD` / `MSTORE` (3 gas each).

Remediation

Storage variables read multiple times inside a function should instead be cached in the memory the first time (costing 1 `SLOAD`) and then read from this cache to avoid multiple `SLOADs`.

Bug ID

SSP_4274_42

Bug Type

STORAGE VARIABLE CACHING IN MEMORY

Severity

● Gas

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L350 - L396

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L350 - L396

```

function quickSwapWithSellTaxToken(
    address _tokenIn,
    address _to,
    uint _afterTax
) external ZeroAddress(_to) {
    // Construct the token swap path
    uint256 _amountIn = IERC20(_tokenIn).balanceOf(msg.sender);
    address[] memory path = new address[](2);
    path[0] = _tokenIn;
    path[1] = WETH;

    IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
    IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);

    IUniswapV2Router02(UNISWAP_V2_ROUTER)
        .swapExactTokensForETHSupportingFeeOnTransferTokens(
            IERC20(_tokenIn).balanceOf(address(this)),
            0,
            path,
            address(this),
            block.timestamp
        );

    (
        ,
        uint256 maintanierFee,
        uint256 platformFee,
        uint256 amountToSend
```

/BuyContract (1).sol L350 - L396

```
            uint256 maintanierFee,
            uint256 platformFee,
            uint256 amountToSend
        ) = percentageCalculation(_afterTax);

        (bool success, ) = maintanierAddress.call{value: maintanierFee}("");
        require(success, "ETH transfer failed To Maintainer");
        success = false;
        (success, ) = platformAddress.call{value: platformFee}("");
        require(success, "ETH transfer failed To Platform");
        success = false;
        (success, ) = _to.call{value: amountToSend}("");
        require(success, "ETH transfer failed To User");

        emit TokensSwapped(
            _tokenIn,
            WETH, // ETH address
            _amountIn,
            amountToSend,
            _to
        );
    }

    /**
```

Description

The contract `BuyContract` is using the state variable `UNISWAP_V2_ROUTER` multiple times in the function `quickSwapWithSellTaxToken`.

`SLOADs` are expensive (100 gas after the 1st one) compared to `MLOAD` / `MSTORE` (3 gas each).

Remediation

Storage variables read multiple times inside a function should instead be cached in the memory the first time (costing 1 `SLOAD`) and then read from this cache to avoid multiple `SLOADs`.

Bug ID	Bug Type	
SSP_4274_42	STORAGE VARIABLE CACHING IN MEMORY	
Severity	Action Taken	Detection Method
● Gas	⚠ *Pending Fix*	Automated

Line No.	File Location
L350 - L396	/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L350 - L396

```

    function quickSwapWithSellTaxToken(
        address _tokenIn,
        address _to,
        uint _afterTax
    ) external ZeroAddress(_to) {
        // Construct the token swap path
        uint256 _amountIn = IERC20(_tokenIn).balanceOf(msg.sender);
        address[] memory path = new address[](2);
        path[0] = _tokenIn;
        path[1] = WETH;

        IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
        IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);

        IUniswapV2Router02(UNISWAP_V2_ROUTER)
            .swapExactTokensForETHSupportingFeeOnTransferTokens(
                IERC20(_tokenIn).balanceOf(address(this)),
                0,
                path,
                address(this),
                block.timestamp
            );

        (
            ,
            uint256 maintanierFee,
            uint256 platformFee,
            uint256 amountToSend
```

/BuyContract (1).sol L350 - L396

```
            uint256 maintanierFee,
            uint256 platformFee,
            uint256 amountToSend
        ) = percentageCalculation(_afterTax);

        (bool success, ) = maintanierAddress.call{value: maintanierFee}("");
        require(success, "ETH transfer failed To Maintainer");
        success = false;
        (success, ) = platformAddress.call{value: platformFee}("");
        require(success, "ETH transfer failed To Platform");
        success = false;
        (success, ) = _to.call{value: amountToSend}("");
        require(success, "ETH transfer failed To User");

        emit TokensSwapped(
            _tokenIn,
            WETH, // ETH address
            _amountIn,
            amountToSend,
            _to
        );
    }

    /**
```

Description

The contract `BuyContract` is using the state variable `WETH` multiple times in the function `quickSwapWithSellTaxToken`.
`SLOADs` are expensive (100 gas after the 1st one) compared to `MLOAD` / `MSTORE` (3 gas each).

Remediation

Storage variables read multiple times inside a function should instead be cached in the memory the first time (costing 1 `SLOAD`) and then read from this cache to avoid multiple `SLOADs`.

Bug ID

SSP_4274_49

Bug Type

STORAGE VARIABLE CACHING IN MEMORY

Severity

● Gas

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L432 - L452

File Location

/BuyContract (1).sol

Affected Code

/BuyContract (1).sol

L432 - L452

```
     */
    function getAmountOutMin(
        address _tokenIn,
        address _tokenOut,
        uint256 _amountIn
    ) external view returns (uint256) {
        // Construct the token swap path
        address[] memory path;
        path = new address[](2);
        if (_tokenIn == WETH) {
            path[0] = WETH;
            path[1] = _tokenOut;
        } else {
            path[0] = _tokenOut;
            path[1] = WETH;
        }

        // Get the minimum amount of token Out
        uint256[] memory amountOutMins = IUniswapV2Router02(UNISWAP_V2_ROUTER)
            .getAmountsOut(_amountIn, path);
        return amountOutMins[path.length - 1];
    }

    function setPlatformAddress(
```

Description

The contract `BuyContract` is using the state variable `WETH` multiple times in the function `getAmountOutMin`. `SLOADs` are expensive (100 gas after the 1st one) compared to `MLOAD` / `MSTORE` (3 gas each).

Remediation

Storage variables read multiple times inside a function should instead be cached in the memory the first time (costing 1 `SLOAD`) and then read from this cache to avoid multiple `SLOADs`.

Bug ID	Bug Type	
SSP_4274_37	SUPERFLUOUS EVENT FIELDS	
Severity	Action Taken	Detection Method
● Gas	⚠ *Pending Fix*	Automated

Line No.	File Location
L110 - L116	/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L110 - L116

```
109
110         emit TokensSwapped(
111             WETH, // ETH address
112             _tokenOut,
113             amountToSend,
114             _amountOut,
115             _to
116         );
117     }
118
```

Description

`block.timestamp` and `block.number` are by default added to event information. Adding them manually costs extra gas.

Remediation

`block.timestamp` and `block.number` do not need to be added manually. Consider removing them from the emitted events.

Bug ID	Bug Type	
SSP_4274_38	SUPERFLUOUS EVENT FIELDS	
Severity	Action Taken	Detection Method
● Gas	⚠ *Pending Fix*	Automated

Line No.	File Location
L273 - L279	/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L273 - L279

```
272
273         emit TokensSwapped(
274             WETH, // ETH address
275             _tokenOut,
276             amountToSend,
277             amount,
278             _to
279         );
280     }
281
```

Description

`block.timestamp` and `block.number` are by default added to event information. Adding them manually costs extr a gas.

Remediation

`block.timestamp` and `block.number` do not need to be added manually. Consider removing them from the emitte d events.

Bug ID	Bug Type	
SSP_4274_50	**UNUSED IMPORTS**	
Severity	Action Taken	Detection Method
● Gas	⚠ *Pending Fix*	Automated

Line No.	File Location
L4 - L4	/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L4 - L4

```
3  import "uniswap-v2-contract/contracts/uniswap-v2-periphery/interfaces/IUniswapV2Router02.sol";
4  import "uniswap-v2-contract/contracts/uniswap-v2-core/interfaces/IUniswapV2Factory.sol";
5  import "uniswap-v2-contract/contracts/uniswap-v2-core/interfaces/IUniswapV2Pair.sol";
6  import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
```

Description

Solidity is a Gas-constrained language. Having unused code or import statements incurs extra gas usage when deploying the contract.
The contract was found to be importing the file uniswap-v2-contract/contracts/uniswap-v2-core/interfaces/IUniswapV2Factory.sol which is not used anywhere in the code.

Remediation

It is recommended to remove the import statement if it’s not supposed to be used.

Bug ID	Bug Type	
SSP_4274_51	**UNUSED IMPORTS**	

Severity	Action Taken	Detection Method
● Gas	⚠ *Pending Fix*	Automated

Line No.	File Location
L5 - L5	/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L5 - L5

```
4   import "uniswap-v2-contract/contracts/uniswap-v2-core/interfaces/IUniswapV2Factory.sol";
5   import "uniswap-v2-contract/contracts/uniswap-v2-core/interfaces/IUniswapV2Pair.sol";
6   import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
7   import "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";
```

Description

Solidity is a Gas-constrained language. Having unused code or import statements incurs extra gas usage when deploying the contract.
The contract was found to be importing the file uniswap-v2-contract/contracts/uniswap-v2-core/interfaces/IUniswapV2Pair.sol which is not used anywhere in the code.

Remediation

It is recommended to remove the import statement if it’s not supposed to be used.

Bug ID	Bug Type	
SSP_4274_52	**UNUSED IMPORTS**	
Severity	Action Taken	Detection Method
● Gas	⚠ *Pending Fix*	Automated

Line No.	File Location
L8 - L8	/BuyContract (1).sol

Affected Code

/BuyContract (1).sol L8 - L8

```
7   import "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";
8   import "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
9
10   contract BuyContract is OwnableUpgradeable {
```

Description

Solidity is a Gas-constrained language. Having unused code or import statements incurs extra gas usage when deploying the contract.

The contract was found to be importing the file @openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol which is not used anywhere in the code.

Remediation

It is recommended to remove the import statement if it’s not supposed to be used.

5. Scan History

● Critical ● High ● Medium ● Low ● Informational ● Gas

No	Date	Security Score	Scan Overview
1.	2024-02-29	76.40	● 6 ● 7 ● 0 ● 12 ● 12 ● 20

6. Dislaimer

The Reports neither endorse nor condemn any specific project or team, nor do they guarantee the security of any specific project. The contents of this report do not, and should not be interpreted as having any bearing on, the economics of tokens, token sales, or any other goods, services, or assets.

The security audit is not meant to replace functional testing done before a software release.

There is no warranty that all possible security issues of a particular smart contract(s) will be found by the tool, i.e., It is not guaranteed that there will not be any further findings based solely on the results of this evaluation.

Emerging technologies such as Smart Contracts and Solidity carry a high level of technical risk and uncertainty. There is no warranty or representation made by this report to any Third Party in regards to the quality of code, the business model or the proprietors of any such business model, or the legal compliance of any business.

In no way should a third party use these reports to make any decisions about buying or selling a token, product, service, or any other asset. It should be noted that this report is not investment advice, is not intended to be relied on as investment advice, and has no endorsement of this project or team. It does not serve as a guarantee as to the project's absolute security.

The assessment provided by CertiFi is subject to dependencies and 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. CertiFi owes no duty to any third party by virtue of publishing these Reports.

As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with several independent manual audits including manual audit and a public bug bounty the program to ensure security of the smart contracts.