



CertifiTM
CRYPTO AGENCY

@certifiagency

Audit Report

3/22/2024

Bot Bros Sniper Bot

www.certifiagency.com

Table of Contents.

01 Vulnerability Classification and Severity

02 Executive Summary

03 Findings Summary

04 Vulnerability Details

- UPGRADEABLE CONTRACT WITHOUT OWNER
- USE OF FLOATING PRAGMA
- OUTDATED COMPILER VERSION
- BLOCK VALUES AS A PROXY FOR TIME
- MISSING PAYABLE IN CALL FUNCTION
- UNUSED RECEIVE FALBACK
- CHEAPER INEQUALITIES IN REQUIRE()
- DEFINE CONSTRUCTOR AS PAYABLE
- FUNCTION SHOULD RETURN STRUCT
- STORAGE VARIABLE CACHING IN MEMORY
- SUPERFLUOUS EVENT FIELDS
- UNUSED IMPORTS
- USE SELFBALANCE() INSTEAD OF ADDRESS(THIS).BALANCE

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.

02. Executive Summary



Bot Sniper Fee Buy

Uploaded Solidity File(s)

Published on 22 Mar 2024

Language

Solidity

Audit Methodology

**Static Scanning
Manual Review**

Website

-

Publishers/Owner Name

-

Organization

Contact Email

-



Security Score is GREAT

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

3. Findings Summary



Bot Sniper Fee Buy

File Scan



Security Score
91.47/100



Scan duration
8 Min



Lines of code
504



3

Crit

0

High

0

Med

2

Low

9

Info

15

Gas



ACTION TAKEN

0

 Fixed

3

 False Positive

0

 Won't Fix

26

 Pending Fix

Bug ID	Severity	Bug Type	Detection Method	Line No	Status
SSP_4516_1	● Critical	UPGRADEABLE CONTRACT WITHOUT OWNER	Automated	L473 - L478	 Addressed
SSP_4516_2	● Critical	UPGRADEABLE CONTRACT WITHOUT OWNER	Automated	L480 - L485	 Addressed
SSP_4516_3	● Critical	UPGRADEABLE CONTRACT WITHOUT OWNER	Automated	L491 - L502	 Addressed
SSP_4516_22	● Low	USE OF FLOATING PRAGMA	Automated	L3 - L3	 Pending Fix
SSP_4516_16	● Low	OUTDATED COMPILER VERSION	Automated	L3 - L3	 Pending Fix
SSP_4516_23	● Informational	BLOCK VALUES AS A PROXY FOR TIME	Automated	L118 - L118	 Pending Fix
SSP_4516_24	● Informational	BLOCK VALUES AS A PROXY FOR TIME	Automated	L165 - L165	 Pending Fix
SSP_4516_24	● Informational	BLOCK VALUES AS A PROXY FOR TIME	Automated	L219 - L219	 Pending Fix
SSP_4516_27	● Informational	BLOCK VALUES AS A PROXY FOR TIME	Automated	L281 - L281	 Pending Fix
SSP_4516_28	● Informational	BLOCK VALUES AS A PROXY FOR TIME	Automated	L332 - L332	 Pending Fix
SSP_4516_28	● Informational	BLOCK VALUES AS A PROXY FOR TIME	Automated	L388 - L388	 Pending Fix
SSP_4516_4	● Informational	MISSING PAYABLE IN CALL FUNCTION	Automated	L175 - L175	 Pending Fix
SSP_4516_5	● Informational	MISSING PAYABLE IN CALL FUNCTION	Automated	L178 - L178	 Pending Fix
SSP_4516_6	● Informational	MISSING PAYABLE IN CALL FUNCTION	Automated	L181 - L181	 Pending Fix
SSP_4516_4	● Informational	MISSING PAYABLE IN CALL FUNCTION	Automated	L229 - L229	 Pending Fix

Bug ID	Severity	Bug Type	Detection Method	Line No	Status
SSP_4516_5	● Informational	MISSING PAYABLE IN CALL FUNCTION	Automated	L232 - L232	Pending Fix
SSP_4516_6	● Informational	MISSING PAYABLE IN CALL FUNCTION	Automated	L235 - L235	Pending Fix
SSP_4516_4	● Informational	MISSING PAYABLE IN CALL FUNCTION	Automated	L342 - L342	Pending Fix
SSP_4516_5	● Informational	MISSING PAYABLE IN CALL FUNCTION	Automated	L345 - L345	Pending Fix
SSP_4516_6	● Informational	MISSING PAYABLE IN CALL FUNCTION	Automated	L348 - L348	Pending Fix
SSP_4516_4	● Informational	MISSING PAYABLE IN CALL FUNCTION	Automated	L398 - L398	Pending Fix
SSP_4516_5	● Informational	MISSING PAYABLE IN CALL FUNCTION	Automated	L401 - L401	Pending Fix
SSP_4516_6	● Informational	MISSING PAYABLE IN CALL FUNCTION	Automated	L404 - L404	Pending Fix
SSP_4516_29	● Informational	MISSING PAYABLE IN CALL FUNCTION	Automated	L500 - L500	Pending Fix
SSP_4516_26	● Informational	UNUSED RECEIVE FALBACK	Automated	L487 - L489	Pending Fix
SSP_4516_13	● Gas	CHEAPER INEQUALITIES IN REQUIRE()	Automated	L497 - L497	Pending Fix
SSP_4516_7	● Gas	DEFINE CONSTRUCTOR AS PAYABLE	Automated	L27 - L29	Pending Fix
SSP_4516_20	● Gas	FUNCTION SHOULD RETURN STRUCT	Automated	L425 - L441	Pending Fix
SSP_4516_8	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L88 - L128	Pending Fix
SSP_4516_9	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L138 - L191	Pending Fix
SSP_4516_9	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L138 - L191	Pending Fix
SSP_4516_10	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L199 - L245	Pending Fix
SSP_4516_10	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L199 - L245	Pending Fix

Bug ID	Severity	Bug Type	Detection Method	Line No	Status
SSP_4516_12	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L254 - L295	 Pending Fix
SSP_4516_12	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L254 - L295	 Pending Fix
SSP_4516_14	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L305 - L358	 Pending Fix
SSP_4516_14	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L305 - L358	 Pending Fix
SSP_4516_15	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L366 - L414	 Pending Fix
SSP_4516_15	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L366 - L414	 Pending Fix
SSP_4516_21	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L450 - L471	 Pending Fix
SSP_4516_17	● Gas	SUPERFLUOUS EVENT FIELDS	Automated	L121 - L127	 Pending Fix
SSP_4516_18	● Gas	SUPERFLUOUS EVENT FIELDS	Automated	L288 - L294	 Pending Fix
SSP_4516_19	● Gas	SUPERFLUOUS EVENT FIELDS	Automated	L501 - L501	 Pending Fix
SSP_4516_25	● Gas	UNUSED IMPORTS	Automated	L8 - L8	 Pending Fix

4. Vulnerability Details

Bug ID: SSP_4516_1 | Bug Type: UPGRADEABLE CONTRACT WITHOUT OWNER

Severity: Critical | Action Taken: Pending Fix | Detection Method: Automated

Line No.: L473 - L478 | File Location: /BuyContract.sol

Affected Code

```
/BuyContract.sol:L473 - L478

472
473     function setPlatformAddress(
474         address _account
475     ) external onlyOwner ZeroAddress(_account) {
476         emit PlatformAddressSet(_platformAddress, _account);
477         _platformAddress = _account;
478     }
479
480     function setMaintainerAddress(
```

Description

When using upgradeable contracts, it is essential to implement an initializer that will call the base contract's initializers in turn. If the initializer is not called, the contract owner stays zero-initialized, meaning using `onlyOwner` will always revert.

Remediation

Consider implementing an initializer to allow the use of `onlyOwner`

Bug ID	Bug Type	
SSP_4516_2	UPGRADEABLE CONTRACT WITHOUT OWNER	
Severity	Action Taken	Detection Method
● Critical	⚠ Pending Fix	Automated

Line No. File Location
L480 - L485 /BuyContract.sol

Affected Code

/BuyContract.sol L480 - L485

```
479
480     function setMaintainerAddress(
481         address _account
482     ) external onlyOwner ZeroAddress(_account) {
483         emit MaintainerAddressSet(_maintainerAddress, _account);
484         _maintainerAddress = _account;
485     }
486
487     receive() external payable {
```

Description

When using upgradeable contracts, it is essential to implement an initializer that will call the base contract's initializers in turn. If the initializer is not called, the contract owner stays zero-initialized, meaning using `onlyOwner` will always revert.

Remediation

Consider implementing an initializer to allow the use of `onlyOwner`

Bug ID	Bug Type	
SSP_4516_3	UPGRADEABLE CONTRACT WITHOUT OWNER	
Severity	Action Taken	Detection Method
● Critical	⚠ Pending Fix	Automated

Line No.	File Location
L491 - L502	/BuyContract.sol

Affected Code

/BuyContract.sol L491 - L502

```
490
491     function withdrawEther(
492         address payable recipient,
493         uint256 amount
494     ) external onlyOwner {
495         require(recipient != address(0), "Invalid recipient address");
496         require(
497             address(this).balance >= amount,
498             "Insufficient balance"
499         );
500         recipient.call{value: amount}("");
501         emit EtherWithdrawn(recipient, amount);
502     }
503 }
```

Description

When using upgradeable contracts, it is essential to implement an initializer that will call the base contract's initializers in turn. If the initializer is not called, the contract owner stays zero-initialized, meaning using `onlyOwner` will always revert.

Remediation

Consider implementing an initializer to allow the use of `onlyOwner`

Bug ID	Bug Type	
SSP_4516_22	USE OF FLOATING PRAGMA	
Severity	Action Taken	Detection Method
● Low	⚠ Pending Fix	Automated

Line No.	File Location
L3 - L3	/BuyContract.sol

Affected Code

/BuyContract.sol L3 - L3

```

2 // pragma solidity ^0.8.23;
3 pragma solidity ^0.8.0;
4 import "uniswap-v2-contract/contracts/uniswap-v2-periphery/interfaces/IUniswapV2Router02.sol";
5 import "@openzeppelin/contracts/token/ERC20/IERC20.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.sol'] - ^0.8.0`

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 them changes in the future.

Instead of `^0.8.0` use `pragma solidity v0.8.24`, which is a stable and recommended version right now.

Bug ID	Bug Type	
SSP_4516_16	OUTDATED COMPILER VERSION	
Severity	Action Taken	Detection Method
● Low	⚠ Pending Fix	Automated
Line No.	File Location	
L3 - L3	/BuyContract.sol	

Affected Code

/BuyContract.sol L3 - L3

```
2 // pragma solidity ^0.8.23;
3 pragma solidity ^0.8.0;
4 import "uniswap-v2-contract/contracts/uniswap-v2-periphery/interfaces/IUniswapV2Router02.sol";
5 import "@openzeppelin/contracts/token/ERC20/IERC20.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.sol'] - ^0.8.0

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 v0.8.24, which patches most solidity vulnerabilities.

Bug ID	Bug Type	
SSP_4516_23	BLOCK VALUES AS A PROXY FOR TIME	
Severity	Action Taken	Detection Method
● Informational	 Pending Fix	Automated

Line No.	File Location
L118 - L118	/BuyContract.sol

Affected Code

```
/BuyContract.sol                                         L118 - L118

117         _to,
118         block.timestamp
119     ) [0];
120
```

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_4516_24	BLOCK VALUES AS A PROXY FOR TIME	
Severity	Action Taken	Detection Method
● Informational	 Pending Fix	Automated

Line No.	File Location
L165 - L165	/BuyContract.sol

Affected Code

```
/BuyContract.sol                                         L165 - L165

164         address(this),
165         block.timestamp
166     )[1];
167
```

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_4516_24	BLOCK VALUES AS A PROXY FOR TIME	
Severity	Action Taken	Detection Method
● Informational	 Pending Fix	Automated

Line No.	File Location
L219 - L219	/BuyContract.sol

Affected Code

```
/BuyContract.sol                                         L219 - L219

218         address(this),
219         block.timestamp
220         )[1];
221
```

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_4516_27	BLOCK VALUES AS A PROXY FOR TIME	
Severity	Action Taken	Detection Method
● Informational	 Pending Fix	Automated

Line No.	File Location
L281 - L281	/BuyContract.sol

Affected Code

```
/BuyContract.sol L281 - L281

280         value: amountToSend
281     }(_amountOutMin, path, _to, block.timestamp);
282
283     uint amount = IUniswapV2Router02(UNISWAP_V2_ROUTER).getAmountsOut(
```

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_4516_28	BLOCK VALUES AS A PROXY FOR TIME	
Severity	Action Taken	Detection Method
● Informational	 Pending Fix	Automated

Line No.	File Location
L332 - L332	/BuyContract.sol

Affected Code

```
/BuyContract.sol                                         L332 - L332

331         address(this),
332         block.timestamp
333     );
334
```

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_4516_28	BLOCK VALUES AS A PROXY FOR TIME	
Severity	Action Taken	Detection Method
● Informational	 Pending Fix	Automated

Line No.	File Location
L388 - L388	/BuyContract.sol

Affected Code

```
/BuyContract.sol                                         L388 - L388

387         address(this),
388         block.timestamp
389     );
390
```

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_4516_4	MISSING PAYABLE IN CALL FUNCTION	
Severity	Action Taken	Detection Method
● Informational	 Pending Fix	Automated

Line No.	File Location
L175 - L175	/BuyContract.sol

Affected Code

/BuyContract.sol L175 - L175

```
174  
175     (bool success, ) = _maintainerAddress.call{value: maintainerFee}("");  
176     require(success, "Maintainer transfer failed");  
177     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	Bug Type	
SSP_4516_5	MISSING PAYABLE IN CALL FUNCTION	
Severity	Action Taken	Detection Method
● Informational	 Pending Fix	Automated

Line No.	File Location
L178 - L178	/BuyContract.sol

Affected Code

/BuyContract.sol L178 - L178

```
177     success = false;
178     (success, ) = _platformAddress.call{value: platformFee}("");
179     require(success, "Platform transfer failed");
180     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	Bug Type	
SSP_4516_6	MISSING PAYABLE IN CALL FUNCTION	
Severity	Action Taken	Detection Method
● Informational	 Pending Fix	Automated

Line No.	File Location
L181 - L181	/BuyContract.sol

Affected Code

```
/BuyContract.sol L181 - L181

180     success = false;
181     (success, ) = _to.call{value: amountToSend}("");
182     require(success, "user transfer failed");
183
```

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	Bug Type	
SSP_4516_4	MISSING PAYABLE IN CALL FUNCTION	
Severity	Action Taken	Detection Method
● Informational	 Pending Fix	Automated

Line No.	File Location
L229 - L229	/BuyContract.sol

Affected Code

/BuyContract.sol L229 - L229

```
228  
229     (bool success, ) = _maintainerAddress.call{value: maintainerFee}("");  
230     require(success, "Maintainer transfer failed");  
231     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_4516_5	MISSING PAYABLE IN CALL FUNCTION	
Severity	Action Taken	Detection Method
● Informational	 Pending Fix	Automated

Line No.	File Location
L232 - L232	/BuyContract.sol

Affected Code

/BuyContract.sol L232 - L232

```
231     success = false;
232     (success, ) = _platformAddress.call{value: platformFee}("");
233     require(success, "Platform transfer failed");
234     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_4516_6	MISSING PAYABLE IN CALL FUNCTION	
Severity	Action Taken	Detection Method
● Informational	 Pending Fix	Automated

Line No.	File Location
L235 - L235	/BuyContract.sol

Affected Code

```
/BuyContract.sol L235 - L235

234     success = false;
235     (success, ) = _to.call{value: amountToSend}("");
236     require(success, "user transfer failed");
237
```

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_4516_4	MISSING PAYABLE IN CALL FUNCTION	
Severity	Action Taken	Detection Method
● Informational	 Pending Fix	Automated

Line No.	File Location
L342 - L342	/BuyContract.sol

Affected Code

/BuyContract.sol L342 - L342

```
341  
342     (bool success, ) = _maintainerAddress.call{value: maintainerFee}("");  
343     require(success, "Maintainer transfer failed");  
344     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_4516_5	MISSING PAYABLE IN CALL FUNCTION	
Severity	Action Taken	Detection Method
● Informational	 Pending Fix	Automated

Line No.	File Location
L345 - L345	/BuyContract.sol

Affected Code

/BuyContract.sol L345 - L345

```
344     success = false;
345     (success, ) = _platformAddress.call{value: platformFee}("");
346     require(success, "Platform transfer failed");
347     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_4516_6	MISSING PAYABLE IN CALL FUNCTION	
Severity	Action Taken	Detection Method
● Informational	 Pending Fix	Automated

Line No.	File Location
L348 - L348	/BuyContract.sol

Affected Code

```
/BuyContract.sol L348 - L348

347     success = false;
348     (success, ) = _to.call{value: amountToSend}("");
349     require(success, "user transfer failed");
350
```

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_4516_4	MISSING PAYABLE IN CALL FUNCTION	
Severity	Action Taken	Detection Method
● Informational	 Pending Fix	Automated

Line No.	File Location
L398 - L398	/BuyContract.sol

Affected Code

/BuyContract.sol L398 - L398

```
397  
398     (bool success, ) = _maintainerAddress.call{value: maintainerFee}("");  
399     require(success, "Maintainer transfer failed");  
400     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_4516_5	MISSING PAYABLE IN CALL FUNCTION	
Severity	Action Taken	Detection Method
● Informational	 Pending Fix	Automated

Line No.	File Location
L401 - L401	/BuyContract.sol

Affected Code

/BuyContract.sol L401 - L401

```
400     success = false;
401     (success, ) = _platformAddress.call{value: platformFee}("");
402     require(success, "Platform transfer failed");
403     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_4516_6	MISSING PAYABLE IN CALL FUNCTION	
Severity	Action Taken	Detection Method
● Informational	 Pending Fix	Automated

Line No.	File Location
L404 - L404	/BuyContract.sol

Affected Code

```
/BuyContract.sol L404 - L404  
403     success = false;  
404     (success, ) = _to.call{value: amountToSend}("");  
405     require(success, "user transfer failed");  
406
```

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_4516_29	MISSING PAYABLE IN CALL FUNCTION	
Severity	Action Taken	Detection Method
● Informational	 Pending Fix	Automated

Line No.	File Location
L500 - L500	/BuyContract.sol

Affected Code

/BuyContract.sol L500 - L500

```
499     );
500     recipient.call{value: amount}("");
501     emit EtherWithdrawn(recipient, amount);
502 }
```

Description

The contract is using a `.call()` method to make external calls along with passing some Ether as `msg.value`. Since the function withdrawEther 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_4516_26	UNUSED RECEIVE FALBACK	
Severity	Action Taken	Detection Method
● Informational	 Pending Fix	Automated

Line No.	File Location
L487 - L489	/BuyContract.sol

Affected Code

/BuyContract.sol L487 - L489

```
486
487     receive() external payable {
488         // React to receiving ether
489     }
490
491     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	Bug Type	
SSP_4516_13	CHEAPER INEQUALITIES IN REQUIRE()	
Severity	Action Taken	Detection Method
● Gas	 Pending Fix	Automated

Line No.	File Location
L497 - L497	/BuyContract.sol

Affected Code

/BuyContract.sol L497 - L497

```
496     require(
497         address(this).balance >= amount,
498         "Insufficient balance"
499     );
```

Description

The contract was found to be performing comparisons using inequalities inside the `require` statement. When inside the `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_4516_7	DEFINE CONSTRUCTOR AS PAYABLE	
Severity	Action Taken	Detection Method
● Gas	 Pending Fix	Automated
Line No.	File Location	
L27 - L29	/BuyContract.sol	

Affected Code

```
/BuyContract.sol L27 - L29  
  
26  
27     constructor() {  
28         _disableInitializers();  
29     }  
30  
31     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	Bug Type	
SSP_4516_20	FUNCTION SHOULD RETURN STRUCT	
Severity	Action Taken	Detection Method
● Gas	 Pending Fix	Automated

Line No.	File Location
L425 - L441	/BuyContract.sol

Affected Code

/BuyContract.sol L425 - L441

```
424
425     function _percentageCalculation(
426         uint256 _amountIn
427     )
428         internal
429         pure
430         returns (
431             uint256 deductionAmount,
432             uint256 maintanierFee,
433             uint256 platformFee,
434             uint256 amountToSwap
435         )
436     {
437         deductionAmount = (_amountIn * 99) / 10000; // 0.99% deduction
438         maintanierFee = (deductionAmount * 40) / 100;
439         platformFee = deductionAmount - maintanierFee;
440         amountToSwap = _amountIn - deductionAmount;
441     }
442
443     /**
444      */

445 }
```

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 readability.

Bug ID	Bug Type	
SSP_4516_8	STORAGE VARIABLE CACHING IN MEMORY	
Severity	Action Taken	Detection Method
● Gas	 Pending Fix	Automated
Line No.	File Location	
L88 - L128	/BuyContract.sol	

Affected Code

/BuyContract.sol L88 - L128

```

87
88     function swapWithFeeBuy(
89         address _tokenOut,
90         uint256 _amountOutMin,
91         address _to
92     ) external payable nonReentrant ZeroAddress(_to) ZeroAmount(_amountOutMin) {
93         require(msg.value != 0, "BC:Invalid ETH Amount");
94         address weth = WETH;
95         // Construct the token swap path
96         (
97             ,
98             uint256 maintanierFee,
99             uint256 platformFee,
100            uint256 amountToSend
101        ) = _percentageCalculation(msg.value);
102        (bool success, ) = _maintanierAddress.call{value: maintanierFee}("");
103        require(success, "Maintainer transfer failed");
104        (success, ) = _platformAddress.call{value: platformFee}("");
105        require(success, "Platform transfer failed");
106
107        address[] memory path;
108
109        path = new address[](2);
110        path[0] = weth;
111        path[1] = _tokenOut;
112
113        uint256 _amountOut = IUniswapV2Router02(UNISWAP_V2_ROUTER)
114            .swapExactETHForTokens{value: amountToSend}(
115                _amountOutMin,
```

```
113     uint256 _amountOut = IUniswapV2Router02(UNISWAP_V2_ROUTER)
114     .swapExactETHForTokens{value: amountToSend}(
115         _amountOutMin,
116         path,
117         _to,
118         block.timestamp
119     )[0];
120
121     emit TokensSwapped(
122         weth, // ETH address
123         _tokenOut,
124         amountToSend,
125         _amountOut,
126         _to
127     );
128 }
129 /**
130 */
```

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_4516_9	STORAGE VARIABLE CACHING IN MEMORY

Severity	Action Taken	Detection Method
● Gas	 Pending Fix	Automated

Line No.	File Location
L138 - L191	/BuyContract.sol

Affected Code

/BuyContract.sol

L138 - L191

```

137
138     function swapWithFeeSell(
139         address _tokenIn,
140         uint256 _amountIn,
141         uint256 _amountOutMin,
142         address _to
143     )
144     external
145     nonReentrant
146     ZeroAddress(_to)
147     ZeroAmount(_amountIn)
148     ZeroAmount(_amountOutMin)
149     {
150         // Construct the token swap path
151
152         address[] memory path = new address[](2);
153         path[0] = _tokenIn;
154         path[1] = WETH;
155
156         TransferHelper.safeTransferFrom(_tokenIn ,msg.sender, address(this), _amountIn );
157         IERC20(_tokenIn).approve(router, _amountIn);
158
159         uint256 amount = IUniswapV2Router02(router)
160             .swapExactTokensForETH(
161                 _amountIn,
162                 _amountOutMin,
163                 path,
164                 address(this),
165                 block.timestamp

```

```
163         path,
164         address(this),
165         block.timestamp
166     )[1];
167
168     (
169     ,
170     uint256 maintanierFee,
171     uint256 platformFee,
172     uint256 amountToSend
173     ) = _percentageCalculation(amount);
174
175     (bool success, ) = _maintanierAddress.call{value: maintanierFee}("");
176     require(success, "Maintainer transfer failed");
177     success = false;
178     (success, ) = _platformAddress.call{value: platformFee}("");
179     require(success, "Platform transfer failed");
180     success = false;
181     (success, ) = _to.call{value: amountToSend}("");
182     require(success, "user transfer failed");
183
184     emit TokensSwapped(
185         _tokenIn,
186         WETH, // ETH address
187         _amountIn,
188         amountToSend,
189         _to
190     );
191 }
192 /**
193 */
```

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	Bug Type
SSP_4516_9	STORAGE VARIABLE CACHING IN MEMORY

Severity	Action Taken	Detection Method
● Gas	 Pending Fix	Automated

Line No.	File Location
L138 - L191	/BuyContract.sol

Affected Code

/BuyContract.sol

L138 - L191

```

137
138     function swapWithFeeSell(
139         address _tokenIn,
140         uint256 _amountIn,
141         uint256 _amountOutMin,
142         address _to
143     )
144     external
145     nonReentrant
146     ZeroAddress(_to)
147     ZeroAmount(_amountIn)
148     ZeroAmount(_amountOutMin)
149     {
150         // Construct the token swap path
151
152         address[] memory path = new address[](2);
153         path[0] = _tokenIn;
154         path[1] = WETH;
155
156         TransferHelper.safeTransferFrom(_tokenIn ,msg.sender, address(this), _amountIn );
157         IERC20(_tokenIn).approve(router, _amountIn);
158
159         uint256 amount = IUniswapV2Router02(router)
160             .swapExactTokensForETH(
161                 _amountIn,
162                 _amountOutMin,
163                 path,
164                 address(this),
165                 block.timestamp

```

```
163         path,
164         address(this),
165         block.timestamp
166     )[1];
167
168     (
169     ,
170     uint256 maintanierFee,
171     uint256 platformFee,
172     uint256 amountToSend
173     ) = _percentageCalculation(amount);
174
175     (bool success, ) = _maintanierAddress.call{value: maintanierFee}("");
176     require(success, "Maintainer transfer failed");
177     success = false;
178     (success, ) = _platformAddress.call{value: platformFee}("");
179     require(success, "Platform transfer failed");
180     success = false;
181     (success, ) = _to.call{value: amountToSend}("");
182     require(success, "user transfer failed");
183
184     emit TokensSwapped(
185         _tokenIn,
186         WETH, // ETH address
187         _amountIn,
188         amountToSend,
189         _to
190     );
191 }
192 /**
193 */
```

Description

The contract `BuyContract` is using the state variable `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_4516_10	STORAGE VARIABLE CACHING IN MEMORY	
Severity	Action Taken	Detection Method
● Gas	 Pending Fix	Automated

Line No. File Location
L199 - L245 /BuyContract.sol

Affected Code

/BuyContract.sol	L199 - L245
------------------	-------------

```

198
199     function quickSwapWithFeeSell(
200         address _tokenIn,
201         address _to
202     ) external nonReentrant ZeroAddress(_to) {
203         // Construct the token swap path
204         address weth = WETH;
205         uint256 _amountIn = IERC20(_tokenIn).balanceOf(msg.sender);
206         address[] memory path = new address[](2);
207         path[0] = _tokenIn;
208         path[1] = weth;
209
210         TransferHelper.safeTransferFrom(_tokenIn ,msg.sender, address(this), _amountIn );
211         IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);
212
213         uint256 amount = IUniswapV2Router02(UNISWAP_V2_ROUTER)
214             .swapExactTokensForETH(
215                 _amountIn,
216                 0,
217                 path,
218                 address(this),
219                 block.timestamp
220             )[1];
221
222         (
223             ,
224             uint256 maintainerFee,
225             uint256 platformFee,
226             uint256 amountToSend

```

```
224     uint256 maintanierFee,
225     uint256 platformFee,
226     uint256 amountToSend
227 ) = _percentageCalculation(amount);
228
229     (bool success, ) = _maintanierAddress.call{value: maintanierFee}("");
230     require(success, "Maintainer transfer failed");
231     success = false;
232     (success, ) = _platformAddress.call{value: platformFee}("");
233     require(success, "Platform transfer failed");
234     success = false;
235     (success, ) = _to.call{value: amountToSend}("");
236     require(success, "user transfer failed");
237
238     emit TokensSwapped(
239         _tokenIn,
240         weth, // ETH address
241         _amountIn,
242         amountToSend,
243         _to
244     );
245 }
246 /**
247 */
```

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	Bug Type	
SSP_4516_10	STORAGE VARIABLE CACHING IN MEMORY	
Severity	Action Taken	Detection Method
● Gas	 Pending Fix	Automated

Line No. File Location
L199 - L245 /BuyContract.sol

Affected Code

/BuyContract.sol	L199 - L245
------------------	-------------

```

198
199     function quickSwapWithFeeSell(
200         address _tokenIn,
201         address _to
202     ) external nonReentrant ZeroAddress(_to) {
203         // Construct the token swap path
204         address weth = WETH;
205         uint256 _amountIn = IERC20(_tokenIn).balanceOf(msg.sender);
206         address[] memory path = new address[](2);
207         path[0] = _tokenIn;
208         path[1] = weth;
209
210         TransferHelper.safeTransferFrom(_tokenIn ,msg.sender, address(this), _amountIn );
211         IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);
212
213         uint256 amount = IUniswapV2Router02(UNISWAP_V2_ROUTER)
214             .swapExactTokensForETH(
215                 _amountIn,
216                 0,
217                 path,
218                 address(this),
219                 block.timestamp
220             )[1];
221
222         (
223             ,
224             uint256 maintainerFee,
225             uint256 platformFee,
226             uint256 amountToSend

```

```
224     uint256 maintanierFee,
225     uint256 platformFee,
226     uint256 amountToSend
227 ) = _percentageCalculation(amount);
228
229     (bool success, ) = _maintanierAddress.call{value: maintanierFee}("");
230     require(success, "Maintainer transfer failed");
231     success = false;
232     (success, ) = _platformAddress.call{value: platformFee}("");
233     require(success, "Platform transfer failed");
234     success = false;
235     (success, ) = _to.call{value: amountToSend}("");
236     require(success, "user transfer failed");
237
238     emit TokensSwapped(
239         _tokenIn,
240         weth, // ETH address
241         _amountIn,
242         amountToSend,
243         _to
244     );
245 }
246
247 /* *
```

Description

The contract `BuyContract` is using the state variable `weth` multiple times in the function `quickSwapWithFeeSel1`.

`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_4516_12	STORAGE VARIABLE CACHING IN MEMORY

Severity	Action Taken	Detection Method
● Gas	 Pending Fix	Automated

Line No.	File Location
L254 - L295	/BuyContract.sol

Affected Code

/BuyContract.sol

L254 - L295

```

253
254     function swapWithBuyTaxToken(
255         address _tokenOut,
256         uint256 _amountOutMin,
257         address _to
258     ) external payable nonReentrant ZeroAddress(_to) ZeroAmount(_amountOutMin) {
259         require(msg.value != 0, "BC:Invalid ETH Amount");
260         // Construct the token swap path
261         address weth = WETH;
262         (
263             ,
264             uint256 maintanierFee,
265             uint256 platformFee,
266             uint256 amountToSend
267         ) = _percentageCalculation(msg.value);
268         (bool success, ) = _maintanierAddress.call{value: maintanierFee}("");
269         require(success, "Maintainer transfer failed");
270         (success, ) = _platformAddress.call{value: platformFee}("");
271         require(success, "Platform transfer failed");
272
273         address[] memory path;
274
275         path = new address[](2);
276         path[0] = weth;
277         path[1] = _tokenOut;
278         IUniswapV2Router02(UNISWAP_V2_ROUTER)
279             .swapExactETHForTokensSupportingFeeOnTransferTokens{
280                 value: amountToSend
281             }(_amountOutMin, path, _to, block.timestamp);

```

```
279         .swapExactETHForTokensSupportingFeeOnTransferTokens{  
280             value: amountToSend  
281         }(_amountOutMin, path, _to, block.timestamp);  
282  
283         uint amount = IUniswapV2Router02(UNISWAP_V2_ROUTER).getAmountsOut(  
284             amountToSend,  
285             path  
286         )[0];  
287  
288         emit TokensSwapped(  
289             weth, // ETH address  
290             _tokenOut,  
291             amountToSend,  
292             amount,  
293             _to  
294         );  
295     }  
296  
297     /* *
```

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	Bug Type
SSP_4516_12	STORAGE VARIABLE CACHING IN MEMORY

Severity	Action Taken	Detection Method
● Gas	 Pending Fix	Automated

Line No.	File Location
L254 - L295	/BuyContract.sol

Affected Code

/BuyContract.sol

L254 - L295

```

253
254     function swapWithBuyTaxToken(
255         address _tokenOut,
256         uint256 _amountOutMin,
257         address _to
258     ) external payable nonReentrant ZeroAddress(_to) ZeroAmount(_amountOutMin) {
259         require(msg.value != 0, "BC:Invalid ETH Amount");
260         // Construct the token swap path
261         address weth = WETH;
262         (
263             ,
264             uint256 maintanierFee,
265             uint256 platformFee,
266             uint256 amountToSend
267         ) = _percentageCalculation(msg.value);
268         (bool success, ) = _maintanierAddress.call{value: maintanierFee}("");
269         require(success, "Maintainer transfer failed");
270         (success, ) = _platformAddress.call{value: platformFee}("");
271         require(success, "Platform transfer failed");
272
273         address[] memory path;
274
275         path = new address[](2);
276         path[0] = weth;
277         path[1] = _tokenOut;
278         IUniswapV2Router02(UNISWAP_V2_ROUTER)
279             .swapExactETHForTokensSupportingFeeOnTransferTokens{
280                 value: amountToSend
281             }(_amountOutMin, path, _to, block.timestamp);

```

```
279     .swapExactETHForTokensSupportingFeeOnTransferTokens{  
280         value: amountToSend  
281     }(_amountOutMin, path, _to, block.timestamp);  
282  
283     uint amount = IUniswapV2Router02(UNISWAP_V2_ROUTER).getAmountsOut(  
284         amountToSend,  
285         path  
286     )[0];  
287  
288     emit TokensSwapped(  
289         weth, // ETH address  
290         _tokenOut,  
291         amountToSend,  
292         amount,  
293         _to  
294     );  
295 }  
296  
297 /* *
```

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	Bug Type	
SSP_4516_14	STORAGE VARIABLE CACHING IN MEMORY	
Severity	Action Taken	Detection Method
● Gas	 Pending Fix	Automated

Line No. File Location
L305 - L358 /BuyContract.sol

Affected Code

/BuyContract.sol	L305 - L358
------------------	-------------

```

304
305     function swapWithSellTaxToken(
306         address _tokenIn,
307         uint256 _amountIn,
308         uint256 _amountOutMin,
309         address _to,
310         uint _afterTax
311     )
312     external
313     nonReentrant
314     ZeroAddress(_to)
315     ZeroAmount(_amountIn)
316     ZeroAmount(_amountOutMin)
317     {
318         // Construct the token swap path
319
320         address[] memory path = new address[](2);
321         path[0] = _tokenIn;
322         path[1] = weth;
323         TransferHelper.safeTransferFrom(_tokenIn ,msg.sender, address(this), _amountIn );
324         IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);
325
326         IUniswapV2Router02(UNISWAP_V2_ROUTER)
327             .swapExactTokensForETHSupportingFeeOnTransferTokens(
328                 IERC20(_tokenIn).balanceOf(address(this)),
329                 _amountOutMin,
330                 path,
331                 address(this),
332                 block.timestamp

```

```
330         path,
331         address(this),
332         block.timestamp
333     );
334
335     (
336     ,
337     uint256 maintanierFee,
338     uint256 platformFee,
339     uint256 amountToSend
340 ) = _percentageCalculation(_afterTax);
341
342     (bool success, ) = _maintanierAddress.call{value: maintanierFee}("");
343     require(success, "Maintainer transfer failed");
344     success = false;
345     (success, ) = _platformAddress.call{value: platformFee}("");
346     require(success, "Platform transfer failed");
347     success = false;
348     (success, ) = _to.call{value: amountToSend}("");
349     require(success, "user transfer failed");
350
351     emit TokensSwapped(
352         _tokenIn,
353         weth, // ETH address
354         _amountIn,
355         amountToSend,
356         _to
357     );
358 }
359 /**
360 */
```

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	Bug Type	
SSP_4516_14	STORAGE VARIABLE CACHING IN MEMORY	
Severity	Action Taken	Detection Method
● Gas	 Pending Fix	Automated

Line No. File Location
L305 - L358 /BuyContract.sol

Affected Code

/BuyContract.sol	L305 - L358
------------------	-------------

```

304
305     function swapWithSellTaxToken(
306         address _tokenIn,
307         uint256 _amountIn,
308         uint256 _amountOutMin,
309         address _to,
310         uint _afterTax
311     )
312     external
313     nonReentrant
314     ZeroAddress(_to)
315     ZeroAmount(_amountIn)
316     ZeroAmount(_amountOutMin)
317     {
318         // Construct the token swap path
319
320         address[] memory path = new address[](2);
321         path[0] = _tokenIn;
322         path[1] = weth;
323         TransferHelper.safeTransferFrom(_tokenIn ,msg.sender, address(this), _amountIn );
324         IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);
325
326         IUniswapV2Router02(UNISWAP_V2_ROUTER)
327             .swapExactTokensForETHSupportingFeeOnTransferTokens(
328                 IERC20(_tokenIn).balanceOf(address(this)),
329                 _amountOutMin,
330                 path,
331                 address(this),
332                 block.timestamp

```

```
330         path,
331         address(this),
332         block.timestamp
333     );
334
335     (
336     ,
337     uint256 maintanierFee,
338     uint256 platformFee,
339     uint256 amountToSend
340 ) = _percentageCalculation(_afterTax);
341
342     (bool success, ) = _maintanierAddress.call{value: maintanierFee}("");
343     require(success, "Maintainer transfer failed");
344     success = false;
345     (success, ) = _platformAddress.call{value: platformFee}("");
346     require(success, "Platform transfer failed");
347     success = false;
348     (success, ) = _to.call{value: amountToSend}("");
349     require(success, "user transfer failed");
350
351     emit TokensSwapped(
352         _tokenIn,
353         weth, // ETH address
354         _amountIn,
355         amountToSend,
356         _to
357     );
358 }
359 /**
360 */
```

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	Bug Type	
SSP_4516_15	STORAGE VARIABLE CACHING IN MEMORY	
Severity	Action Taken	Detection Method
● Gas	 Pending Fix	Automated

Line No. File Location
L366 - L414 /BuyContract.sol

Affected Code

/BuyContract.sol	L366 - L414
------------------	-------------

```

365
366     function quickSwapWithSellTaxToken(
367         address _tokenIn,
368         address _to,
369         uint _afterTax
370     ) external nonReentrant ZeroAddress(_to) {
371         // Construct the token swap path
372         address router = UNISWAP_V2_ROUTER;
373         address weth = WETH;
374         uint256 _amountIn = IERC20(_tokenIn).balanceOf(msg.sender);
375         address[] memory path = new address[](2);
376         path[0] = _tokenIn;
377         path[1] = weth;
378
379         TransferHelper.safeTransferFrom(_tokenIn ,msg.sender, address(this), _amountIn );
380         IERC20(_tokenIn).approve(router, _amountIn);
381
382         IUniswapV2Router02(router)
383             .swapExactTokensForETHSupportingFeeOnTransferTokens(
384                 IERC20(_tokenIn).balanceOf(address(this)),
385                 0,
386                 path,
387                 address(this),
388                 block.timestamp
389             );
390
391         (
392             /
393             uint256 maintanierFee,

```

```
391      (
392      ,
393      uint256 maintanierFee,
394      uint256 platformFee,
395      uint256 amountToSend
396      ) = _percentageCalculation(_afterTax);
397
398      (bool success, ) = _maintanierAddress.call{value: maintanierFee}("");
399      require(success, "Maintainer transfer failed");
400      success = false;
401      (success, ) = _platformAddress.call{value: platformFee}("");
402      require(success, "Platform transfer failed");
403      success = false;
404      (success, ) = _to.call{value: amountToSend}("");
405      require(success, "user transfer failed");
406
407      emit TokensSwapped(
408          _tokenIn,
409          weth, // ETH address
410          _amountIn,
411          amountToSend,
412          _to
413      );
414  }
415
416  /* *
```

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	Bug Type	
SSP_4516_15	STORAGE VARIABLE CACHING IN MEMORY	
Severity	Action Taken	Detection Method
● Gas	 Pending Fix	Automated

Line No. File Location
L366 - L414 /BuyContract.sol

Affected Code

/BuyContract.sol	L366 - L414
------------------	-------------

```

365
366     function quickSwapWithSellTaxToken(
367         address _tokenIn,
368         address _to,
369         uint _afterTax
370     ) external nonReentrant ZeroAddress(_to) {
371         // Construct the token swap path
372         address router = UNISWAP_V2_ROUTER;
373         address weth = WETH;
374         uint256 _amountIn = IERC20(_tokenIn).balanceOf(msg.sender);
375         address[] memory path = new address[](2);
376         path[0] = _tokenIn;
377         path[1] = weth;
378
379         TransferHelper.safeTransferFrom(_tokenIn ,msg.sender, address(this), _amountIn );
380         IERC20(_tokenIn).approve(router, _amountIn);
381
382         IUniswapV2Router02(router)
383             .swapExactTokensForETHSupportingFeeOnTransferTokens(
384                 IERC20(_tokenIn).balanceOf(address(this)),
385                 0,
386                 path,
387                 address(this),
388                 block.timestamp
389             );
390
391         (
392             /
393             uint256 maintanierFee,

```

```
391     (
392     ,
393     uint256 maintanierFee,
394     uint256 platformFee,
395     uint256 amountToSend
396     ) = _percentageCalculation(_afterTax);
397
398     (bool success, ) = _maintanierAddress.call{value: maintanierFee}("");
399     require(success, "Maintainer transfer failed");
400     success = false;
401     (success, ) = _platformAddress.call{value: platformFee}("");
402     require(success, "Platform transfer failed");
403     success = false;
404     (success, ) = _to.call{value: amountToSend}("");
405     require(success, "user transfer failed");
406
407     emit TokensSwapped(
408         _tokenIn,
409         weth, // ETH address
410         _amountIn,
411         amountToSend,
412         _to
413     );
414 }
415
416 /* *
```

Description

The contract `BuyContract` is using the state variable `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_4516_21	STORAGE VARIABLE CACHING IN MEMORY	
Severity	Action Taken	Detection Method
● Gas	 Pending Fix	Automated
Line No.	File Location	
L450 - L471	/BuyContract.sol	

Affected Code

/BuyContract.sol L450 - L471

```

449     */
450     function getAmountOutMin(
451         address _tokenIn,
452         address _tokenOut,
453         uint256 _amountIn
454     ) external view returns (uint256) {
455         // Construct the token swap path
456         address weth = WETH;
457         address[] memory path;
458         path = new address[](2);
459         if (_tokenIn == weth) {
460             path[0] = weth;
461             path[1] = _tokenOut;
462         } else {
463             path[0] = _tokenOut;
464             path[1] = weth;
465         }
466
467         // Get the minimum amount of token Out
468         uint256[] memory amountOutMins = IUniswapV2Router02(UNISWAP_V2_ROUTER)
469             .getAmountsOut(_amountIn, path);
470         return amountOutMins[path.length - 1];
471     }
472
473     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_4516_17	SUPERFLUOUS EVENT FIELDS	
Severity	Action Taken	Detection Method
● Gas	 Pending Fix	Automated

Line No.	File Location
L121 - L127	/BuyContract.sol

Affected Code

/BuyContract.sol L121 - L127

```
120
121     emit TokensSwapped(
122         weth, // ETH address
123         _tokenOut,
124         amountToSend,
125         _amountOut,
126         _to
127     );
128 }
129
```

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_4516_18	SUPERFLUOUS EVENT FIELDS	
Severity	Action Taken	Detection Method
● Gas	 Pending Fix	Automated

Line No.	File Location
L288 - L294	/BuyContract.sol

Affected Code

/BuyContract.sol L288 - L294

```
287
288     emit TokensSwapped(
289         weth, // ETH address
290         _tokenOut,
291         amountToSend,
292         amount,
293         _to
294     );
295 }
296
```

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_4516_19	SUPERFLUOUS EVENT FIELDS	
Severity	Action Taken	Detection Method
● Gas	 Pending Fix	Automated

Line No.	File Location
L501 - L501	/BuyContract.sol

Affected Code

/BuyContract.sol L501 - L501

```
500     recipient.call{value: amount}("");
501     emit EtherWithdrawn(recipient, amount);
502 }
503 }
```

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_4516_25	UNUSED IMPORTS	
Severity	Action Taken	Detection Method
● Gas	 Pending Fix	Automated

Line No.	File Location
L8 - L8	/BuyContract.sol

Affected Code

/BuyContract.sol L8 - L8

```
7 import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
8 import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
9 import "libraries/TransferHelper.sol";
10
```

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/token/ERC20/utils/SafeERC20.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_4516_11	USE SELFBALANCE() INSTEAD OF ADDRESS(this).BALANCE	
Severity	Action Taken	Detection Method
● Gas	 Pending Fix	Automated
Line No.	File Location	
L497 - L497	/BuyContract.sol	

Affected Code

/BuyContract.sol L497 - L497

```
496     require(
497         address(this).balance >= amount,
498         "Insufficient balance"
499     );
```

Description

In Solidity, efficient use of gas is paramount to ensure cost-effective execution on the Ethereum blockchain. Gas can be optimized when obtaining contract balance by using `selfbalance()` rather than `address(this).balance` because it bypasses gas costs and refunds, which are not required for obtaining the contract's balance.

Remediation

To rectify this issue, developers are encouraged to replace instances of `address(this).balance` with `selfbalance()` wherever applicable. This optimization not only ensures streamlined gas operations but also contributes to substantial cost savings during contract execution.

5. Scan History

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

No	Date	Security Score	Scan Overview					
1.	2024-03-22	91.47	● 3	● 0	● 0	● 2	● 9	● 15

6. Disclaimer

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 Agency 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 Agency 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 program to ensure the security of the smart contracts.