CertiFi™
CRYPTO AGENCY
@certifiagency

Audit Report

February 28, 2024

BnB No Fee Contract Audit Report

www.certifiagency.com

Table of Contents.

- SUPERFLUOUS EVENT FIELDS
- UNUSED IMPORTS

05 Scan History

06 Disclaimer

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. Executive Summary

BnBNoFee-Contract

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 BnBNoFee-Contract using CertiFi 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 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

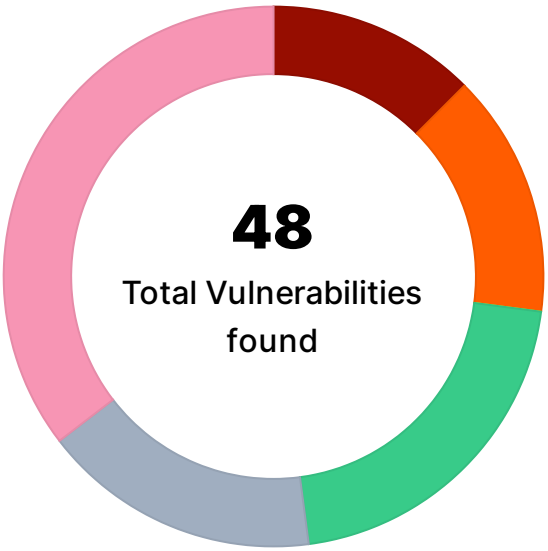
The CertiFi Team recommends running regular audit scans to identify any vulnerabilities that are introduced after BnBNoFee-Contract introduces new features or refactors the code.

3. Findings Summary

BnBNoFee-Contract

File Scan

Security Score	Scan duration	Lines of code
70.90/100	**1 secs**	**354**



6	7	0	10	8	17
Crit	High	Med	Low	Info	Gas

ACTION TAKEN

0	0	0	48
Fixed	False Positive	Won't Fix	Pending Fix

Bug ID	Severity	Bug Type	Detection Method	Line No	Status
SSP_4275_17	Critical	INCORRECT ACCESS CONTROL	Automated	L62 - L90	Pending Fix
SSP_4275_18	Critical	INCORRECT ACCESS CONTROL	Automated	L100 - L138	Pending Fix
SSP_4275_19	Critical	INCORRECT ACCESS CONTROL	Automated	L146 - L178	Pending Fix
SSP_4275_20	Critical	INCORRECT ACCESS CONTROL	Automated	L187 - L216	Pending Fix
SSP_4275_21	Critical	INCORRECT ACCESS CONTROL	Automated	L226 - L265	Pending Fix
SSP_4275_22	Critical	INCORRECT ACCESS CONTROL	Automated	L273 - L306	Pending Fix
SSP_4275_31	High	REENTRANCY	Automated	L62 - L90	Pending Fix
SSP_4275_32	High	REENTRANCY	Automated	L100 - L138	Pending Fix
SSP_4275_33	High	REENTRANCY	Automated	L146 - L178	Pending Fix
SSP_4275_34	High	REENTRANCY	Automated	L187 - L216	Pending Fix
SSP_4275_35	High	REENTRANCY	Automated	L226 - L265	Pending Fix
SSP_4275_36	High	REENTRANCY	Automated	L273 - L306	Pending Fix
SSP_4275_5	High	UNCHECKED TRANSFER	Automated	L116 - L116	Pending Fix
SSP_4275_5	High	UNCHECKED TRANSFER	Automated	L156 - L156	Pending Fix
SSP_4275_5	High	UNCHECKED TRANSFER	Automated	L243 - L243	Pending Fix

Bug ID	Severity	Bug Type	Detection Method	Line No	Status
SSP_4275_5	High	UNCHECKED TRANSFER	Automated	L284 - L284	*Pending Fix*
SSP_4275_37	Low	EVENT BASED REENTRANCY	Automated	L62 - L90	*Pending Fix*
SSP_4275_38	Low	EVENT BASED REENTRANCY	Automated	L100 - L138	*Pending Fix*
SSP_4275_39	Low	EVENT BASED REENTRANCY	Automated	L146 - L178	*Pending Fix*
SSP_4275_40	Low	EVENT BASED REENTRANCY	Automated	L187 - L216	*Pending Fix*
SSP_4275_41	Low	EVENT BASED REENTRANCY	Automated	L226 - L265	*Pending Fix*
SSP_4275_42	Low	EVENT BASED REENTRANCY	Automated	L273 - L306	*Pending Fix*
SSP_4275_11	Low	USE OF FLOATING PRAGMA	Automated	L2 - L2	*Pending Fix*
SSP_4275_6	Low	MISSING EVENTS	Automated	L341 - L352	*Pending Fix*
SSP_4275_47	Low	OUTDATED COMPILER VERSION	Automated	L2 - L2	*Pending Fix*
SSP_4275_4	Low	USE OWNABLE2STEP	Automated	L10 - L10	*Pending Fix*
SSP_4275_14	Informational	BLOCK VALUES AS A PROXY FOR TIME	Automated	L80 - L80	*Pending Fix*
SSP_4275_15	Informational	BLOCK VALUES AS A PROXY FOR TIME	Automated	L125 - L125	*Pending Fix*
SSP_4275_15	Informational	BLOCK VALUES AS A PROXY FOR TIME	Automated	L165 - L165	*Pending Fix*
SSP_4275_27	Informational	BLOCK VALUES AS A PROXY FOR TIME	Automated	L203 - L203	*Pending Fix*
SSP_4275_28	Informational	BLOCK VALUES AS A PROXY FOR TIME	Automated	L252 - L252	*Pending Fix*
SSP_4275_28	Informational	BLOCK VALUES AS A PROXY FOR TIME	Automated	L293 - L293	*Pending Fix*
SSP_4275_1	Informational	MISSING PAYABLE IN CALL FUNCTION	Automated	L128 - L128	*Pending Fix*

Bug ID	Severity	Bug Type	Detection Method	Line No	Status
SSP_4275_1	Informational	MISSING PAYABLE IN CALL FUNCTION	Automated	L168 - L168	*Pending Fix*
SSP_4275_2	Informational	MISSING PAYABLE IN CALL FUNCTION	Automated	L255 - L255	*Pending Fix*
SSP_4275_2	Informational	MISSING PAYABLE IN CALL FUNCTION	Automated	L296 - L296	*Pending Fix*
SSP_4275_12	Informational	UNUSED RECEIVE FALLBACK	Automated	L337 - L339	*Pending Fix*
SSP_4275_23	Informational	USE CALL INSTEAD OF TRANSFER OR SEND	Automated	L351 - L351	*Pending Fix*
SSP_4275_48	Gas	CHEAPER CONDITIONAL OPERATORS	Automated	L67 - L67	*Pending Fix*
SSP_4275_48	Gas	CHEAPER CONDITIONAL OPERATORS	Automated	L192 - L192	*Pending Fix*
SSP_4275_7	Gas	CHEAPER INEQUALITIES IN REQUIRE()	Automated	L347 - L347	*Pending Fix*
SSP_4275_3	Gas	DEFINE CONSTRUCTOR AS PAYABLE	Automated	L18 - L20	*Pending Fix*
SSP_4275_10	Gas	LONG REQUIRE/REVERT STRINGS	Automated	L346 - L349	*Pending Fix*
SSP_4275_8	Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L62 - L90	*Pending Fix*
SSP_4275_9	Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L100 - L138	*Pending Fix*
SSP_4275_9	Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L100 - L138	*Pending Fix*
SSP_4275_13	Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L146 - L178	*Pending Fix*
SSP_4275_13	Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L146 - L178	*Pending Fix*
SSP_4275_16	Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L187 - L216	*Pending Fix*
SSP_4275_16	Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L187 - L216	*Pending Fix*
SSP_4275_43	Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L226 - L265	*Pending Fix*

Bug ID	Severity	Bug Type	Detection Method	Line No	Status
SSP_4275_43	Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L226 - L265	*Pending Fix*
SSP_4275_44	Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L273 - L306	*Pending Fix*
SSP_4275_44	Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L273 - L306	*Pending Fix*
SSP_4275_45	Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L315 - L335	*Pending Fix*
SSP_4275_29	Gas	SUPERFLUOUS EVENT FIELDS	Automated	L83 - L89	*Pending Fix*
SSP_4275_30	Gas	SUPERFLUOUS EVENT FIELDS	Automated	L131 - L137	*Pending Fix*
SSP_4275_30	Gas	SUPERFLUOUS EVENT FIELDS	Automated	L171 - L177	*Pending Fix*
SSP_4275_46	Gas	SUPERFLUOUS EVENT FIELDS	Automated	L209 - L215	*Pending Fix*
SSP_4275_24	Gas	UNUSED IMPORTS	Automated	L4 - L4	*Pending Fix*
SSP_4275_25	Gas	UNUSED IMPORTS	Automated	L5 - L5	*Pending Fix*

4. Vulnerability Details

Bug ID	Bug Type
SSP_4275_17	INCORRECT ACCESS CONTROL

Severity	Action Taken	Detection Method
Critical	*Pending Fix*	Automated

Line No.	File Location
L62 - L90	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L62 - L90

```
61
62      function swapWithFeeBuy(
63          address _tokenOut,
64          uint256 _amountOutMin,
65          address _to
66      ) external payable ZeroAddress(_to) ZeroAmount(_amountOutMin) {
67          require(msg.value > 0, "BC:Invalid ETH Amount");
68          // Construct the token swap path
69          address[] memory path;
70
71          path = new address[](2);
72          path[0] = WETH;
73          path[1] = _tokenOut;
74
75          uint256 _amountOut = IUniswapV2Router02(UNISWAP_V2_ROUTER)
76              .swapExactETHForTokens{value: msg.value}(
77              _amountOutMin,
78              path,
79              _to,
80              block.timestamp
81          )[0];
82
83          emit TokensSwapped(
84              WETH, // ETH address
85              _tokenOut,
86              msg.value,
87              _amountOut,
88              _to
89          );
```

/BuyContract (2).sol L62 - L90

```
87              _amountOut,
88              _to
89          );
90      }
91
92      /**
```

Description

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

The contract BuyContract is importing an access control library @openzeppelin/contracts-upgradeable/access/Ownab leUpgradeable.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 th ey contain sensitive administrative actions, it is advised to add a suitable modifier to the same

Bug ID	Bug Type
SSP_4275_18	INCORRECT ACCESS CONTROL

Severity	Action Taken	Detection Method
Critical	*Pending Fix*	Automated

Line No.	File Location
L100 - L138	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L100 - L138

```
99
100     function swapWithFeeSell(
101         address _tokenIn,
102         uint256 _amountIn,
103         uint256 _amountOutMin,
104         address _to
105     )
106         external
107         ZeroAddress(_to)
108         ZeroAmount(_amountIn)
109         ZeroAmount(_amountOutMin)
110     {
111         // Construct the token swap path
112         address[] memory path = new address[](2);
113         path[0] = _tokenIn;
114         path[1] = WETH;
115
116         IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
117         IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);
118
119         uint256 amount = IUniswapV2Router02(UNISWAP_V2_ROUTER)
120             .swapExactTokensForETH(
121                 _amountIn,
122                 _amountOutMin,
123                 path,
124                 address(this),
125                 block.timestamp
126             )[1];
127
```

/BuyContract (2).sol L100 - L138

```
125                 block.timestamp
126             )[1];
127
128         (bool success, ) = _to.call{value: amount}("");
129         require(success, "ETH transfer failed To User");
130
131         emit TokensSwapped(
132             _tokenIn,
133             WETH, // ETH address
134             _amountIn,
135             amount,
136             _to
137         );
138     }
139
140     /**
```

Description

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

The contract BuyContract is importing an access control library @openzeppelin/contracts-upgradeable/access/Ownab leUpgradeable.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 th ey contain sensitive administrative actions, it is advised to add a suitable modifier to the same

Bug ID	Bug Type
SSP_4275_19	INCORRECT ACCESS CONTROL

Severity	Action Taken	Detection Method
Critical	*Pending Fix*	Automated

Line No.	File Location
L146 - L178	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L146 - L178

```
145
146     function quickSwapWithFeeSell(
147         address _tokenIn,
148         address _to
149     ) external ZeroAddress(_to) {
150         // Construct the token swap path
151         uint256 _amountIn = IERC20(_tokenIn).balanceOf(msg.sender);
152         address[] memory path = new address[](2);
153         path[0] = _tokenIn;
154         path[1] = WETH;
155
156         IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
157         IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);
158
159         uint256 amount = IUniswapV2Router02(UNISWAP_V2_ROUTER)
160             .swapExactTokensForETH(
161                 _amountIn,
162                 0,
163                 path,
164                 address(this),
165                 block.timestamp
166             )[1];
167
168         (bool success, ) = _to.call{value: amount}("");
169         require(success, "ETH transfer failed To User");
170
171         emit TokensSwapped(
172             _tokenIn,
173             WETH, // ETH address
```

/BuyContract (2).sol L146 - L178

```
171         emit TokensSwapped(
172             _tokenIn,
173             WETH, // ETH address
174             _amountIn,
175             amount,
176             _to
177         );
178     }
179
180     /**
```

Description

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

The contract BuyContract is importing an access control library @openzeppelin/contracts-upgradeable/access/Ownab leUpgradeable.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 th ey contain sensitive administrative actions, it is advised to add a suitable modifier to the same

Bug ID	Bug Type
SSP_4275_20	INCORRECT ACCESS CONTROL

Severity	Action Taken	Detection Method
Critical	*Pending Fix*	Automated

Line No.	File Location
L187 - L216	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L187 - L216

```
186
187     function swapWithBuyTaxToken(
188         address _tokenOut,
189         uint256 _amountOutMin,
190         address _to
191     ) external payable ZeroAddress(_to) ZeroAmount(_amountOutMin) {
192         require(msg.value > 0, "BC:Invalid ETH Amount");
193         // Construct the token swap path
194         address[] memory path;
195
196         path = new address[](2);
197         path[0] = WETH;
198         path[1] = _tokenOut;
199
200         IUniswapV2Router02(UNISWAP_V2_ROUTER)
201             .swapExactETHForTokensSupportingFeeOnTransferTokens{
202             value: msg.value
203         }(_amountOutMin, path, _to, block.timestamp);
204         uint amount = IUniswapV2Router02(UNISWAP_V2_ROUTER).getAmountsOut(
205             msg.value,
206             path
207         )[0];
208
209         emit TokensSwapped(
210             WETH, // ETH address
211             _tokenOut,
212             msg.value,
213             amount,
214             _to
```

/BuyContract (2).sol L187 - L216

```
212             msg.value,
213             amount,
214             _to
215         );
216     }
217
218     /**
```

Description

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

The contract BuyContract is importing an access control library @openzeppelin/contracts-upgradeable/access/Ownab leUpgradeable.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 th ey contain sensitive administrative actions, it is advised to add a suitable modifier to the same

Bug ID	Bug Type
SSP_4275_21	INCORRECT ACCESS CONTROL

Severity	Action Taken	Detection Method
Critical	*Pending Fix*	Automated

Line No.	File Location
L226 - L265	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L226 - L265

```
225
226     function swapWithSellTaxToken(
227         address _tokenIn,
228         uint256 _amountIn,
229         uint256 _amountOutMin,
230         address _to,
231         uint _afterTax
232     )
233         external
234         ZeroAddress(_to)
235         ZeroAmount(_amountIn)
236         ZeroAmount(_amountOutMin)
237     {
238         // Construct the token swap path
239         address[] memory path = new address[](2);
240         path[0] = _tokenIn;
241         path[1] = WETH;
242
243         IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
244         IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);
245
246         IUniswapV2Router02(UNISWAP_V2_ROUTER)
247             .swapExactTokensForETHSupportingFeeOnTransferTokens(
248                 IERC20(_tokenIn).balanceOf(address(this)),
249                 _amountOutMin,
250                 path,
251                 address(this),
252                 block.timestamp
253             );
```

/BuyContract (2).sol L226 - L265

```
251                 address(this),
252                 block.timestamp
253             );
254
255         (bool success, ) = _to.call{value: _afterTax}("");
256         require(success, "ETH transfer failed To User");
257
258         emit TokensSwapped(
259             _tokenIn,
260             WETH, // ETH address
261             _amountIn,
262             _afterTax,
263             _to
264         );
265     }
266
267     /**
```

Description

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

The contract BuyContract is importing an access control library @openzeppelin/contracts-upgradeable/access/Ownab leUpgradeable.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 th ey contain sensitive administrative actions, it is advised to add a suitable modifier to the same

Bug ID	Bug Type
SSP_4275_22	INCORRECT ACCESS CONTROL

Severity	Action Taken	Detection Method
Critical	*Pending Fix*	Automated

Line No.	File Location
L273 - L306	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L273 - L306

```
272
273     function quickSwapWithSellTaxToken(
274         address _tokenIn,
275         address _to,
276         uint _afterTax
277     ) external ZeroAddress(_to) {
278         // Construct the token swap path
279         uint256 _amountIn = IERC20(_tokenIn).balanceOf(msg.sender);
280         address[] memory path = new address[](2);
281         path[0] = _tokenIn;
282         path[1] = WETH;
283
284         IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
285         IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);
286
287         IUniswapV2Router02(UNISWAP_V2_ROUTER)
288             .swapExactTokensForETHSupportingFeeOnTransferTokens(
289                 IERC20(_tokenIn).balanceOf(address(this)),
290                 0,
291                 path,
292                 address(this),
293                 block.timestamp
294             );
295
296         (bool success, ) = _to.call{value: _afterTax}("");
297         require(success, "ETH transfer failed To User");
298
299         emit TokensSwapped(
300             _tokenIn,
```

/BuyContract (2).sol L273 - L306

```
298
299         emit TokensSwapped(
300             _tokenIn,
301             WETH, // ETH address
302             _amountIn,
303             _afterTax,
304             _to
305         );
306     }
307
308     /**
```

Description

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

The contract BuyContract is importing an access control library @openzeppelin/contracts-upgradeable/access/Ownab leUpgradeable.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 th ey contain sensitive administrative actions, it is advised to add a suitable modifier to the same

Bug ID	Bug Type
SSP_4275_31	REENTRANCY

Severity	Action Taken	Detection Method
High	*Pending Fix*	Automated

Line No.	File Location
L62 - L90	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L62 - L90

```
61
62      function swapWithFeeBuy(
63          address _tokenOut,
64          uint256 _amountOutMin,
65          address _to
66      ) external payable ZeroAddress(_to) ZeroAmount(_amountOutMin) {
67          require(msg.value > 0, "BC:Invalid ETH Amount");
68          // Construct the token swap path
69          address[] memory path;
70
71          path = new address[](2);
72          path[0] = WETH;
73          path[1] = _tokenOut;
74
75          uint256 _amountOut = IUniswapV2Router02(UNISWAP_V2_ROUTER)
76              .swapExactETHForTokens{value: msg.value}(
77              _amountOutMin,
78              path,
79              _to,
80              block.timestamp
81          )[0];
82
83          emit TokensSwapped(
84              WETH, // ETH address
85              _tokenOut,
86              msg.value,
87              _amountOut,
88              _to
89          );
```

/BuyContract (2).sol L62 - L90

```
87              _amountOut,
88              _to
89          );
90      }
91
92      /**
```

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.
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 Che cks-Effects-Interactions pattern. The external calls should be executed at the end of the function and all the state-cha nging must happen before the call.

Bug ID	Bug Type
SSP_4275_32	REENTRANCY

Severity	Action Taken	Detection Method
High	*Pending Fix*	Automated

Line No.	File Location
L100 - L138	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L100 - L138

```
99
100     function swapWithFeeSell(
101         address _tokenIn,
102         uint256 _amountIn,
103         uint256 _amountOutMin,
104         address _to
105     )
106         external
107         ZeroAddress(_to)
108         ZeroAmount(_amountIn)
109         ZeroAmount(_amountOutMin)
110     {
111         // Construct the token swap path
112         address[] memory path = new address[](2);
113         path[0] = _tokenIn;
114         path[1] = WETH;
115
116         IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
117         IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);
118
119         uint256 amount = IUniswapV2Router02(UNISWAP_V2_ROUTER)
120             .swapExactTokensForETH(
121                 _amountIn,
122                 _amountOutMin,
123                 path,
124                 address(this),
125                 block.timestamp
126             )[1];
127
```

/BuyContract (2).sol L100 - L138

```
125                 block.timestamp
126             )[1];
127
128         (bool success, ) = _to.call{value: amount}("");
129         require(success, "ETH transfer failed To User");
130
131         emit TokensSwapped(
132             _tokenIn,
133             WETH, // ETH address
134             _amountIn,
135             amount,
136             _to
137         );
138     }
139
140     /**
```

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.
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 Che cks-Effects-Interactions pattern. The external calls should be executed at the end of the function and all the state-cha nging must happen before the call.

Bug ID	Bug Type
SSP_4275_33	REENTRANCY

Severity	Action Taken	Detection Method
High	*Pending Fix*	Automated

Line No.	File Location
L146 - L178	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L146 - L178

```
145
146     function quickSwapWithFeeSell(
147         address _tokenIn,
148         address _to
149     ) external ZeroAddress(_to) {
150         // Construct the token swap path
151         uint256 _amountIn = IERC20(_tokenIn).balanceOf(msg.sender);
152         address[] memory path = new address[](2);
153         path[0] = _tokenIn;
154         path[1] = WETH;
155
156         IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
157         IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);
158
159         uint256 amount = IUniswapV2Router02(UNISWAP_V2_ROUTER)
160             .swapExactTokensForETH(
161                 _amountIn,
162                 0,
163                 path,
164                 address(this),
165                 block.timestamp
166             )[1];
167
168         (bool success, ) = _to.call{value: amount}("");
169         require(success, "ETH transfer failed To User");
170
171         emit TokensSwapped(
172             _tokenIn,
173             WETH, // ETH address
```

/BuyContract (2).sol L146 - L178

```
171         emit TokensSwapped(
172             _tokenIn,
173             WETH, // ETH address
174             _amountIn,
175             amount,
176             _to
177         );
178     }
179
180     /**
```

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.
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 Che cks-Effects-Interactions pattern. The external calls should be executed at the end of the function and all the state-cha nging must happen before the call.

Bug ID	Bug Type
SSP_4275_34	REENTRANCY

Severity	Action Taken	Detection Method
High	*Pending Fix*	Automated

Line No.	File Location
L187 - L216	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L187 - L216

```
186
187     function swapWithBuyTaxToken(
188         address _tokenOut,
189         uint256 _amountOutMin,
190         address _to
191     ) external payable ZeroAddress(_to) ZeroAmount(_amountOutMin) {
192         require(msg.value > 0, "BC:Invalid ETH Amount");
193         // Construct the token swap path
194         address[] memory path;
195
196         path = new address[](2);
197         path[0] = WETH;
198         path[1] = _tokenOut;
199
200         IUniswapV2Router02(UNISWAP_V2_ROUTER)
201             .swapExactETHForTokensSupportingFeeOnTransferTokens{
202             value: msg.value
203         }(_amountOutMin, path, _to, block.timestamp);
204         uint amount = IUniswapV2Router02(UNISWAP_V2_ROUTER).getAmountsOut(
205             msg.value,
206             path
207         )[0];
208
209         emit TokensSwapped(
210             WETH, // ETH address
211             _tokenOut,
212             msg.value,
213             amount,
214             _to
```

/BuyContract (2).sol L187 - L216

```
212             msg.value,
213             amount,
214             _to
215         );
216     }
217
218     /**
```

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.
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 Che cks-Effects-Interactions pattern. The external calls should be executed at the end of the function and all the state-cha nging must happen before the call.

Bug ID	Bug Type
SSP_4275_35	REENTRANCY

Severity	Action Taken	Detection Method
High	*Pending Fix*	Automated

Line No.	File Location
L226 - L265	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L226 - L265

```
225
226     function swapWithSellTaxToken(
227         address _tokenIn,
228         uint256 _amountIn,
229         uint256 _amountOutMin,
230         address _to,
231         uint _afterTax
232     )
233         external
234         ZeroAddress(_to)
235         ZeroAmount(_amountIn)
236         ZeroAmount(_amountOutMin)
237     {
238         // Construct the token swap path
239         address[] memory path = new address[](2);
240         path[0] = _tokenIn;
241         path[1] = WETH;
242
243         IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
244         IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);
245
246         IUniswapV2Router02(UNISWAP_V2_ROUTER)
247             .swapExactTokensForETHSupportingFeeOnTransferTokens(
248                 IERC20(_tokenIn).balanceOf(address(this)),
249                 _amountOutMin,
250                 path,
251                 address(this),
252                 block.timestamp
253             );
```

/BuyContract (2).sol L226 - L265

```
251                 address(this),
252                 block.timestamp
253             );
254
255         (bool success, ) = _to.call{value: _afterTax}("");
256         require(success, "ETH transfer failed To User");
257
258         emit TokensSwapped(
259             _tokenIn,
260             WETH, // ETH address
261             _amountIn,
262             _afterTax,
263             _to
264         );
265     }
266
267     /**
```

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.
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 Che cks-Effects-Interactions pattern. The external calls should be executed at the end of the function and all the state-cha nging must happen before the call.

Bug ID	Bug Type
SSP_4275_36	REENTRANCY

Severity	Action Taken	Detection Method
High	*Pending Fix*	Automated

Line No.	File Location
L273 - L306	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L273 - L306

```
272
273     function quickSwapWithSellTaxToken(
274         address _tokenIn,
275         address _to,
276         uint _afterTax
277     ) external ZeroAddress(_to) {
278         // Construct the token swap path
279         uint256 _amountIn = IERC20(_tokenIn).balanceOf(msg.sender);
280         address[] memory path = new address[](2);
281         path[0] = _tokenIn;
282         path[1] = WETH;
283
284         IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
285         IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);
286
287         IUniswapV2Router02(UNISWAP_V2_ROUTER)
288             .swapExactTokensForETHSupportingFeeOnTransferTokens(
289                 IERC20(_tokenIn).balanceOf(address(this)),
290                 0,
291                 path,
292                 address(this),
293                 block.timestamp
294             );
295
296         (bool success, ) = _to.call{value: _afterTax}("");
297         require(success, "ETH transfer failed To User");
298
299         emit TokensSwapped(
300             _tokenIn,
```

/BuyContract (2).sol L273 - L306

```
298
299         emit TokensSwapped(
300             _tokenIn,
301             WETH, // ETH address
302             _amountIn,
303             _afterTax,
304             _to
305         );
306     }
307
308     /**
```

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.
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 Che cks-Effects-Interactions pattern. The external calls should be executed at the end of the function and all the state-cha nging must happen before the call.

Bug ID	Bug Type
SSP_4275_5	UNCHECKED TRANSFER

Severity	Action Taken	Detection Method
High	*Pending Fix*	Automated

Line No.	File Location
L116 - L116	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L116 - L116

```
115
116         IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
117         IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);
118
```

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	Bug Type
SSP_4275_5	UNCHECKED TRANSFER

Severity	Action Taken	Detection Method
High	*Pending Fix*	Automated

Line No.	File Location
L156 - L156	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L156 - L156

```
155
156         IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
157         IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);
158
```

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	Bug Type
SSP_4275_5	UNCHECKED TRANSFER

Severity	Action Taken	Detection Method
High	*Pending Fix*	Automated

Line No.	File Location
L243 - L243	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L243 - L243

```
242
243         IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
244         IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);
245
```

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	Bug Type
SSP_4275_5	UNCHECKED TRANSFER

Severity	Action Taken	Detection Method
High	*Pending Fix*	Automated

Line No.	File Location
L284 - L284	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L284 - L284

```
283
284         IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
285         IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);
286
```

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	Bug Type
SSP_4275_37	EVENT BASED REENTRANCY

Severity	Action Taken	Detection Method
Low	*Pending Fix*	Automated

Line No.	File Location
L62 - L90	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L62 - L90

```
61
62      function swapWithFeeBuy(
63          address _tokenOut,
64          uint256 _amountOutMin,
65          address _to
66      ) external payable ZeroAddress(_to) ZeroAmount(_amountOutMin) {
67          require(msg.value > 0, "BC:Invalid ETH Amount");
68          // Construct the token swap path
69          address[] memory path;
70
71          path = new address[](2);
72          path[0] = WETH;
73          path[1] = _tokenOut;
74
75          uint256 _amountOut = IUniswapV2Router02(UNISWAP_V2_ROUTER)
76              .swapExactETHForTokens{value: msg.value}(
77              _amountOutMin,
78              path,
79              _to,
80              block.timestamp
81          )[0];
82
83          emit TokensSwapped(
84              WETH, // ETH address
85              _tokenOut,
86              msg.value,
87              _amountOut,
88              _to
89          );
```

/BuyContract (2).sol L62 - L90

```
87              _amountOut,
88              _to
89          );
90      }
91
92      /**
```

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	Bug Type
SSP_4275_38	EVENT BASED REENTRANCY

Severity	Action Taken	Detection Method
Low	*Pending Fix*	Automated

Line No.	File Location
L100 - L138	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L100 - L138

```
99
100     function swapWithFeeSell(
101         address _tokenIn,
102         uint256 _amountIn,
103         uint256 _amountOutMin,
104         address _to
105     )
106         external
107         ZeroAddress(_to)
108         ZeroAmount(_amountIn)
109         ZeroAmount(_amountOutMin)
110     {
111         // Construct the token swap path
112         address[] memory path = new address[](2);
113         path[0] = _tokenIn;
114         path[1] = WETH;
115
116         IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
117         IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);
118
119         uint256 amount = IUniswapV2Router02(UNISWAP_V2_ROUTER)
120             .swapExactTokensForETH(
121                 _amountIn,
122                 _amountOutMin,
123                 path,
124                 address(this),
125                 block.timestamp
126             )[1];
127
```

/BuyContract (2).sol L100 - L138

```
125                 block.timestamp
126             )[1];
127
128         (bool success, ) = _to.call{value: amount}("");
129         require(success, "ETH transfer failed To User");
130
131         emit TokensSwapped(
132             _tokenIn,
133             WETH, // ETH address
134             _amountIn,
135             amount,
136             _to
137         );
138     }
139
140     /**
```

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	Bug Type
SSP_4275_39	EVENT BASED REENTRANCY

Severity	Action Taken	Detection Method
Low	*Pending Fix*	Automated

Line No.	File Location
L146 - L178	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L146 - L178

```
145
146     function quickSwapWithFeeSell(
147         address _tokenIn,
148         address _to
149     ) external ZeroAddress(_to) {
150         // Construct the token swap path
151         uint256 _amountIn = IERC20(_tokenIn).balanceOf(msg.sender);
152         address[] memory path = new address[](2);
153         path[0] = _tokenIn;
154         path[1] = WETH;
155
156         IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
157         IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);
158
159         uint256 amount = IUniswapV2Router02(UNISWAP_V2_ROUTER)
160             .swapExactTokensForETH(
161                 _amountIn,
162                 0,
163                 path,
164                 address(this),
165                 block.timestamp
166             )[1];
167
168         (bool success, ) = _to.call{value: amount}("");
169         require(success, "ETH transfer failed To User");
170
171         emit TokensSwapped(
172             _tokenIn,
173             WETH, // ETH address
```

/BuyContract (2).sol L146 - L178

```
171         emit TokensSwapped(
172             _tokenIn,
173             WETH, // ETH address
174             _amountIn,
175             amount,
176             _to
177         );
178     }
179
180     /**
```

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	Bug Type
SSP_4275_40	EVENT BASED REENTRANCY

Severity	Action Taken	Detection Method
Low	*Pending Fix*	Automated

Line No.	File Location
L187 - L216	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L187 - L216

```
186
187     function swapWithBuyTaxToken(
188         address _tokenOut,
189         uint256 _amountOutMin,
190         address _to
191     ) external payable ZeroAddress(_to) ZeroAmount(_amountOutMin) {
192         require(msg.value > 0, "BC:Invalid ETH Amount");
193         // Construct the token swap path
194         address[] memory path;
195
196         path = new address[](2);
197         path[0] = WETH;
198         path[1] = _tokenOut;
199
200         IUniswapV2Router02(UNISWAP_V2_ROUTER)
201             .swapExactETHForTokensSupportingFeeOnTransferTokens{
202             value: msg.value
203         }(_amountOutMin, path, _to, block.timestamp);
204         uint amount = IUniswapV2Router02(UNISWAP_V2_ROUTER).getAmountsOut(
205             msg.value,
206             path
207         )[0];
208
209         emit TokensSwapped(
210             WETH, // ETH address
211             _tokenOut,
212             msg.value,
213             amount,
214             _to
```

/BuyContract (2).sol L187 - L216

```
212             msg.value,
213             amount,
214             _to
215         );
216     }
217
218     /**
```

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	Bug Type
SSP_4275_41	EVENT BASED REENTRANCY

Severity	Action Taken	Detection Method
Low	*Pending Fix*	Automated

Line No.	File Location
L226 - L265	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L226 - L265

```
225
226     function swapWithSellTaxToken(
227         address _tokenIn,
228         uint256 _amountIn,
229         uint256 _amountOutMin,
230         address _to,
231         uint _afterTax
232     )
233         external
234         ZeroAddress(_to)
235         ZeroAmount(_amountIn)
236         ZeroAmount(_amountOutMin)
237     {
238         // Construct the token swap path
239         address[] memory path = new address[](2);
240         path[0] = _tokenIn;
241         path[1] = WETH;
242
243         IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
244         IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);
245
246         IUniswapV2Router02(UNISWAP_V2_ROUTER)
247             .swapExactTokensForETHSupportingFeeOnTransferTokens(
248                 IERC20(_tokenIn).balanceOf(address(this)),
249                 _amountOutMin,
250                 path,
251                 address(this),
252                 block.timestamp
253             );
```

/BuyContract (2).sol L226 - L265

```
251                 address(this),
252                 block.timestamp
253             );
254
255         (bool success, ) = _to.call{value: _afterTax}("");
256         require(success, "ETH transfer failed To User");
257
258         emit TokensSwapped(
259             _tokenIn,
260             WETH, // ETH address
261             _amountIn,
262             _afterTax,
263             _to
264         );
265     }
266
267     /**
```

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	Bug Type
SSP_4275_42	EVENT BASED REENTRANCY

Severity	Action Taken	Detection Method
Low	*Pending Fix*	Automated

Line No.	File Location
L273 - L306	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L273 - L306

```
272
273     function quickSwapWithSellTaxToken(
274         address _tokenIn,
275         address _to,
276         uint _afterTax
277     ) external ZeroAddress(_to) {
278         // Construct the token swap path
279         uint256 _amountIn = IERC20(_tokenIn).balanceOf(msg.sender);
280         address[] memory path = new address[](2);
281         path[0] = _tokenIn;
282         path[1] = WETH;
283
284         IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
285         IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);
286
287         IUniswapV2Router02(UNISWAP_V2_ROUTER)
288             .swapExactTokensForETHSupportingFeeOnTransferTokens(
289                 IERC20(_tokenIn).balanceOf(address(this)),
290                 0,
291                 path,
292                 address(this),
293                 block.timestamp
294             );
295
296         (bool success, ) = _to.call{value: _afterTax}("");
297         require(success, "ETH transfer failed To User");
298
299         emit TokensSwapped(
300             _tokenIn,
```

/BuyContract (2).sol L273 - L306

```
298
299         emit TokensSwapped(
300             _tokenIn,
301             WETH, // ETH address
302             _amountIn,
303             _afterTax,
304             _to
305         );
306     }
307
308     /**
```

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	Bug Type
SSP_4275_11	USE OF FLOATING PRAGMA

Severity	Action Taken	Detection Method
Low	*Pending Fix*	Automated

Line No.	File Location
L2 - L2	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L2 - L2

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity ^0.8.7;
3  import "uniswap-v2-contract/contracts/uniswap-v2-periphery/interfaces/IUniswapV2Router02.sol";
4  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 (2).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_4275_6	MISSING EVENTS

Severity	Action Taken	Detection Method
Low	*Pending Fix*	Automated

Line No.	File Location
L341 - L352	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L341 - L352

```
340
341     function withdrawEther(
342         address payable recipient,
343         uint256 amount
344     ) external onlyOwner {
345         require(recipient != address(0), "Invalid recipient address");
346         require(
347             address(this).balance >= amount,
348             "Insufficient balance in the contract"
349         );
350
351         recipient.transfer(amount);
352     }
353 }
354
```

Description

Events are inheritable members of contracts. When you call them, they cause the arguments to be stored in the transact ion'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 dif ficult 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_4275_47	OUTDATED COMPILER VERSION

Severity	Action Taken	Detection Method
Low	*Pending Fix*	Automated

Line No.	File Location
L2 - L2	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L2 - L2

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity ^0.8.7;
3  import "uniswap-v2-contract/contracts/uniswap-v2-periphery/interfaces/IUniswapV2Router02.sol";
4  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 af fect the current compiler version.
The following outdated versions were detected:
`['/BuyContract (2).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 sho uld 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	Bug Type
SSP_4275_4	USE OWNABLE2STEP

Severity	Action Taken	Detection Method
Low	*Pending Fix*	Automated

Line No.	File Location
L10 - L10	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L10 - L10

```
9
10  contract BuyContract is OwnableUpgradeable {
11      // Address of the Uniswap v2 router
12      address public UNISWAP_V2_ROUTER;
```

Description

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

Remediation

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

Bug ID	Bug Type
SSP_4275_14	BLOCK VALUES AS A PROXY FOR TIME

Severity	Action Taken	Detection Method
Informational	*Pending Fix*	Automated

Line No.	File Location
L80 - L80	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L80 - L80

```
79              _to,
80              block.timestamp
81          )[0];
82
```

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_4275_15	BLOCK VALUES AS A PROXY FOR TIME

Severity	Action Taken	Detection Method
Informational	*Pending Fix*	Automated

Line No.	File Location
L125 - L125	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L125 - L125

```
124                 address(this),
125                 block.timestamp
126             )[1];
127
```

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_4275_15	BLOCK VALUES AS A PROXY FOR TIME

Severity	Action Taken	Detection Method
Informational	*Pending Fix*	Automated

Line No.	File Location
L165 - L165	/BuyContract (2).sol

Affected Code

/BuyContract (2).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_4275_27	BLOCK VALUES AS A PROXY FOR TIME

Severity	Action Taken	Detection Method
Informational	*Pending Fix*	Automated

Line No.	File Location
L203 - L203	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L203 - L203

```
202             value: msg.value
203         }(_amountOutMin, path, _to, block.timestamp);
204         uint amount = IUniswapV2Router02(UNISWAP_V2_ROUTER).getAmountsOut(
205             msg.value,
```

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_4275_28	BLOCK VALUES AS A PROXY FOR TIME

Severity	Action Taken	Detection Method
Informational	*Pending Fix*	Automated

Line No.	File Location
L252 - L252	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L252 - L252

```
251                 address(this),
252                 block.timestamp
253             );
254
```

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_4275_28	BLOCK VALUES AS A PROXY FOR TIME

Severity	Action Taken	Detection Method
Informational	*Pending Fix*	Automated

Line No.	File Location
L293 - L293	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L293 - L293

```
292                 address(this),
293                 block.timestamp
294             );
295
```

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_4275_1	MISSING PAYABLE IN CALL FUNCTION

Severity	Action Taken	Detection Method
Informational	*Pending Fix*	Automated

Line No.	File Location
L128 - L128	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L128 - L128

```
127
128         (bool success, ) = _to.call{value: amount}("");
129         require(success, "ETH transfer failed To User");
130
```

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_4275_1	MISSING PAYABLE IN CALL FUNCTION

Severity	Action Taken	Detection Method
Informational	*Pending Fix*	Automated

Line No.	File Location
L168 - L168	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L168 - L168

```
167
168         (bool success, ) = _to.call{value: amount}("");
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 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_4275_2	MISSING PAYABLE IN CALL FUNCTION

Severity	Action Taken	Detection Method
Informational	*Pending Fix*	Automated

Line No.	File Location
L255 - L255	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L255 - L255

```
254
255         (bool success, ) = _to.call{value: _afterTax}("");
256         require(success, "ETH transfer failed To User");
257
```

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_4275_2	MISSING PAYABLE IN CALL FUNCTION

Severity	Action Taken	Detection Method
Informational	*Pending Fix*	Automated

Line No.	File Location
L296 - L296	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L296 - L296

```
295
296         (bool success, ) = _to.call{value: _afterTax}("");
297         require(success, "ETH transfer failed To User");
298
```

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_4275_12	UNUSED RECEIVE FALLBACK

Severity	Action Taken	Detection Method
Informational	*Pending Fix*	Automated

Line No.	File Location
L337 - L339	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L337 - L339

```
336
337     receive() external payable {
338         // React to receiving ether
339     }
340
341     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_4275_23	USE CALL INSTEAD OF TRANSFER OR SEND

Severity	Action Taken	Detection Method
Informational	*Pending Fix*	Automated

Line No.	File Location
L351 - L351	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L351 - L351

```
350
351             recipient.transfer(amount);
352         }
353     }
```

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_4275_48	CHEAPER CONDITIONAL OPERATORS

Severity	Action Taken	Detection Method
Gas	*Pending Fix*	Automated

Line No.	File Location
L67 - L67	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L67 - L67

```
66      ) external payable ZeroAddress(_to) ZeroAmount(_amountOutMin) {
67          require(msg.value > 0, "BC:Invalid ETH Amount");
68          // Construct the token swap path
69          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_4275_48	CHEAPER CONDITIONAL OPERATORS

Severity	Action Taken	Detection Method
Gas	*Pending Fix*	Automated

Line No.	File Location
L192 - L192	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L192 - L192

```
191     ) external payable ZeroAddress(_to) ZeroAmount(_amountOutMin) {
192         require(msg.value > 0, "BC:Invalid ETH Amount");
193         // Construct the token swap path
194         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_4275_7	CHEAPER INEQUALITIES IN REQUIRE()

Severity	Action Taken	Detection Method
Gas	*Pending Fix*	Automated

Line No.	File Location
L347 - L347	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L347 - L347

```
346         require(
347             address(this).balance >= amount,
348             "Insufficient balance in the contract"
349         );
```

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_4275_3	DEFINE CONSTRUCTOR AS PAYABLE

Severity	Action Taken	Detection Method
Gas	*Pending Fix*	Automated

Line No.	File Location
L18 - L20	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L18 - L20

```
17
18      constructor() {
19          _disableInitializers();
20      }
21
22      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_4275_10	LONG REQUIRE/REVERT STRINGS

Severity	Action Taken	Detection Method
Gas	*Pending Fix*	Automated

Line No.	File Location
L346 - L349	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L346 - L349

```
345         require(recipient != address(0), "Invalid recipient address");
346         require(
347             address(this).balance >= amount,
348             "Insufficient balance in the contract"
349         );
350
351         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	Bug Type
SSP_4275_8	STORAGE VARIABLE CACHING IN MEMORY

Severity	Action Taken	Detection Method
Gas	*Pending Fix*	Automated

Line No.	File Location
L62 - L90	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L62 - L90

```
61
62      function swapWithFeeBuy(
63          address _tokenOut,
64          uint256 _amountOutMin,
65          address _to
66      ) external payable ZeroAddress(_to) ZeroAmount(_amountOutMin) {
67          require(msg.value > 0, "BC:Invalid ETH Amount");
68          // Construct the token swap path
69          address[] memory path;
70
71          path = new address[](2);
72          path[0] = WETH;
73          path[1] = _tokenOut;
74
75          uint256 _amountOut = IUniswapV2Router02(UNISWAP_V2_ROUTER)
76              .swapExactETHForTokens{value: msg.value}(
77              _amountOutMin,
78              path,
79              _to,
80              block.timestamp
81          )[0];
82
83          emit TokensSwapped(
84              WETH, // ETH address
85              _tokenOut,
86              msg.value,
87              _amountOut,
88              _to
89          );
```

/BuyContract (2).sol L62 - L90

```
87              _amountOut,
88              _to
89          );
90      }
91
92      /**
```

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_4275_9	STORAGE VARIABLE CACHING IN MEMORY

Severity	Action Taken	Detection Method
Gas	*Pending Fix*	Automated

Line No.	File Location
L100 - L138	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L100 - L138

```
99
100     function swapWithFeeSell(
101         address _tokenIn,
102         uint256 _amountIn,
103         uint256 _amountOutMin,
104         address _to
105     )
106         external
107         ZeroAddress(_to)
108         ZeroAmount(_amountIn)
109         ZeroAmount(_amountOutMin)
110     {
111         // Construct the token swap path
112         address[] memory path = new address[](2);
113         path[0] = _tokenIn;
114         path[1] = WETH;
115
116         IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
117         IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);
118
119         uint256 amount = IUniswapV2Router02(UNISWAP_V2_ROUTER)
120             .swapExactTokensForETH(
121                 _amountIn,
122                 _amountOutMin,
123                 path,
124                 address(this),
125                 block.timestamp
126             )[1];
127
```

/BuyContract (2).sol L100 - L138

```
125                 block.timestamp
126             )[1];
127
128         (bool success, ) = _to.call{value: amount}("");
129         require(success, "ETH transfer failed To User");
130
131         emit TokensSwapped(
132             _tokenIn,
133             WETH, // ETH address
134             _amountIn,
135             amount,
136             _to
137         );
138     }
139
140     /**
```

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_4275_9	STORAGE VARIABLE CACHING IN MEMORY

Severity	Action Taken	Detection Method
Gas	*Pending Fix*	Automated

Line No.	File Location
L100 - L138	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L100 - L138

```
99
100     function swapWithFeeSell(
101         address _tokenIn,
102         uint256 _amountIn,
103         uint256 _amountOutMin,
104         address _to
105     )
106         external
107         ZeroAddress(_to)
108         ZeroAmount(_amountIn)
109         ZeroAmount(_amountOutMin)
110     {
111         // Construct the token swap path
112         address[] memory path = new address[](2);
113         path[0] = _tokenIn;
114         path[1] = WETH;
115
116         IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
117         IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);
118
119         uint256 amount = IUniswapV2Router02(UNISWAP_V2_ROUTER)
120             .swapExactTokensForETH(
121                 _amountIn,
122                 _amountOutMin,
123                 path,
124                 address(this),
125                 block.timestamp
126             )[1];
127
```

/BuyContract (2).sol L100 - L138

```
125                 block.timestamp
126             )[1];
127
128         (bool success, ) = _to.call{value: amount}("");
129         require(success, "ETH transfer failed To User");
130
131         emit TokensSwapped(
132             _tokenIn,
133             WETH, // ETH address
134             _amountIn,
135             amount,
136             _to
137         );
138     }
139
140     /**
```

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_4275_13	STORAGE VARIABLE CACHING IN MEMORY

Severity	Action Taken	Detection Method
Gas	*Pending Fix*	Automated

Line No.	File Location
L146 - L178	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L146 - L178

```
145
146     function quickSwapWithFeeSell(
147         address _tokenIn,
148         address _to
149     ) external ZeroAddress(_to) {
150         // Construct the token swap path
151         uint256 _amountIn = IERC20(_tokenIn).balanceOf(msg.sender);
152         address[] memory path = new address[](2);
153         path[0] = _tokenIn;
154         path[1] = WETH;
155
156         IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
157         IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);
158
159         uint256 amount = IUniswapV2Router02(UNISWAP_V2_ROUTER)
160             .swapExactTokensForETH(
161                 _amountIn,
162                 0,
163                 path,
164                 address(this),
165                 block.timestamp
166             )[1];
167
168         (bool success, ) = _to.call{value: amount}("");
169         require(success, "ETH transfer failed To User");
170
171         emit TokensSwapped(
172             _tokenIn,
173             WETH, // ETH address
```

/BuyContract (2).sol L146 - L178

```
171         emit TokensSwapped(
172             _tokenIn,
173             WETH, // ETH address
174             _amountIn,
175             amount,
176             _to
177         );
178     }
179
180     /**
```

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_4275_13	STORAGE VARIABLE CACHING IN MEMORY

Severity	Action Taken	Detection Method
Gas	*Pending Fix*	Automated

Line No.	File Location
L146 - L178	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L146 - L178

```
145
146     function quickSwapWithFeeSell(
147         address _tokenIn,
148         address _to
149     ) external ZeroAddress(_to) {
150         // Construct the token swap path
151         uint256 _amountIn = IERC20(_tokenIn).balanceOf(msg.sender);
152         address[] memory path = new address[](2);
153         path[0] = _tokenIn;
154         path[1] = WETH;
155
156         IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
157         IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);
158
159         uint256 amount = IUniswapV2Router02(UNISWAP_V2_ROUTER)
160             .swapExactTokensForETH(
161                 _amountIn,
162                 0,
163                 path,
164                 address(this),
165                 block.timestamp
166             )[1];
167
168         (bool success, ) = _to.call{value: amount}("");
169         require(success, "ETH transfer failed To User");
170
171         emit TokensSwapped(
172             _tokenIn,
173             WETH, // ETH address
```

/BuyContract (2).sol L146 - L178

```
171         emit TokensSwapped(
172             _tokenIn,
173             WETH, // ETH address
174             _amountIn,
175             amount,
176             _to
177         );
178     }
179
180     /**
```

Description

The contract `BuyContract` is using the state variable `WETH` multiple times in the function `quickSwapWithFeeSel l`.
`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_4275_16	STORAGE VARIABLE CACHING IN MEMORY

Severity	Action Taken	Detection Method
Gas	*Pending Fix*	Automated

Line No.	File Location
L187 - L216	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L187 - L216

```
186
187     function swapWithBuyTaxToken(
188         address _tokenOut,
189         uint256 _amountOutMin,
190         address _to
191     ) external payable ZeroAddress(_to) ZeroAmount(_amountOutMin) {
192         require(msg.value > 0, "BC:Invalid ETH Amount");
193         // Construct the token swap path
194         address[] memory path;
195
196         path = new address[](2);
197         path[0] = WETH;
198         path[1] = _tokenOut;
199
200         IUniswapV2Router02(UNISWAP_V2_ROUTER)
201             .swapExactETHForTokensSupportingFeeOnTransferTokens{
202             value: msg.value
203         }(_amountOutMin, path, _to, block.timestamp);
204         uint amount = IUniswapV2Router02(UNISWAP_V2_ROUTER).getAmountsOut(
205             msg.value,
206             path
207         )[0];
208
209         emit TokensSwapped(
210             WETH, // ETH address
211             _tokenOut,
212             msg.value,
213             amount,
214             _to
```

/BuyContract (2).sol L187 - L216

```
212             msg.value,
213             amount,
214             _to
215         );
216     }
217
218     /**
```

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_4275_16	STORAGE VARIABLE CACHING IN MEMORY

Severity	Action Taken	Detection Method
Gas	*Pending Fix*	Automated

Line No.	File Location
L187 - L216	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L187 - L216

```
186
187     function swapWithBuyTaxToken(
188         address _tokenOut,
189         uint256 _amountOutMin,
190         address _to
191     ) external payable ZeroAddress(_to) ZeroAmount(_amountOutMin) {
192         require(msg.value > 0, "BC:Invalid ETH Amount");
193         // Construct the token swap path
194         address[] memory path;
195
196         path = new address[](2);
197         path[0] = WETH;
198         path[1] = _tokenOut;
199
200         IUniswapV2Router02(UNISWAP_V2_ROUTER)
201             .swapExactETHForTokensSupportingFeeOnTransferTokens{
202             value: msg.value
203         }(_amountOutMin, path, _to, block.timestamp);
204         uint amount = IUniswapV2Router02(UNISWAP_V2_ROUTER).getAmountsOut(
205             msg.value,
206             path
207         )[0];
208
209         emit TokensSwapped(
210             WETH, // ETH address
211             _tokenOut,
212             msg.value,
213             amount,
214             _to
```

/BuyContract (2).sol L187 - L216

```
212             msg.value,
213             amount,
214             _to
215         );
216     }
217
218     /**
```

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_4275_43	STORAGE VARIABLE CACHING IN MEMORY

Severity	Action Taken	Detection Method
Gas	*Pending Fix*	Automated

Line No.	File Location
L226 - L265	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L226 - L265

```
225
226     function swapWithSellTaxToken(
227         address _tokenIn,
228         uint256 _amountIn,
229         uint256 _amountOutMin,
230         address _to,
231         uint _afterTax
232     )
233         external
234         ZeroAddress(_to)
235         ZeroAmount(_amountIn)
236         ZeroAmount(_amountOutMin)
237     {
238         // Construct the token swap path
239         address[] memory path = new address[](2);
240         path[0] = _tokenIn;
241         path[1] = WETH;
242
243         IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
244         IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);
245
246         IUniswapV2Router02(UNISWAP_V2_ROUTER)
247             .swapExactTokensForETHSupportingFeeOnTransferTokens(
248                 IERC20(_tokenIn).balanceOf(address(this)),
249                 _amountOutMin,
250                 path,
251                 address(this),
252                 block.timestamp
253             );
```

/BuyContract (2).sol L226 - L265

```
251                 address(this),
252                 block.timestamp
253             );
254
255         (bool success, ) = _to.call{value: _afterTax}("");
256         require(success, "ETH transfer failed To User");
257
258         emit TokensSwapped(
259             _tokenIn,
260             WETH, // ETH address
261             _amountIn,
262             _afterTax,
263             _to
264         );
265     }
266
267     /**
```

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_4275_43	STORAGE VARIABLE CACHING IN MEMORY

Severity	Action Taken	Detection Method
Gas	*Pending Fix*	Automated

Line No.	File Location
L226 - L265	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L226 - L265

```
225
226     function swapWithSellTaxToken(
227         address _tokenIn,
228         uint256 _amountIn,
229         uint256 _amountOutMin,
230         address _to,
231         uint _afterTax
232     )
233         external
234         ZeroAddress(_to)
235         ZeroAmount(_amountIn)
236         ZeroAmount(_amountOutMin)
237     {
238         // Construct the token swap path
239         address[] memory path = new address[](2);
240         path[0] = _tokenIn;
241         path[1] = WETH;
242
243         IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
244         IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);
245
246         IUniswapV2Router02(UNISWAP_V2_ROUTER)
247             .swapExactTokensForETHSupportingFeeOnTransferTokens(
248                 IERC20(_tokenIn).balanceOf(address(this)),
249                 _amountOutMin,
250                 path,
251                 address(this),
252                 block.timestamp
253             );
```

/BuyContract (2).sol L226 - L265

```
251                 address(this),
252                 block.timestamp
253             );
254
255         (bool success, ) = _to.call{value: _afterTax}("");
256         require(success, "ETH transfer failed To User");
257
258         emit TokensSwapped(
259             _tokenIn,
260             WETH, // ETH address
261             _amountIn,
262             _afterTax,
263             _to
264         );
265     }
266
267     /**
```

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_4275_44	STORAGE VARIABLE CACHING IN MEMORY

Severity	Action Taken	Detection Method
Gas	*Pending Fix*	Automated

Line No.	File Location
L273 - L306	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L273 - L306

```
272
273     function quickSwapWithSellTaxToken(
274         address _tokenIn,
275         address _to,
276         uint _afterTax
277     ) external ZeroAddress(_to) {
278         // Construct the token swap path
279         uint256 _amountIn = IERC20(_tokenIn).balanceOf(msg.sender);
280         address[] memory path = new address[](2);
281         path[0] = _tokenIn;
282         path[1] = WETH;
283
284         IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
285         IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);
286
287         IUniswapV2Router02(UNISWAP_V2_ROUTER)
288             .swapExactTokensForETHSupportingFeeOnTransferTokens(
289                 IERC20(_tokenIn).balanceOf(address(this)),
290                 0,
291                 path,
292                 address(this),
293                 block.timestamp
294             );
295
296         (bool success, ) = _to.call{value: _afterTax}("");
297         require(success, "ETH transfer failed To User");
298
299         emit TokensSwapped(
300             _tokenIn,
```

/BuyContract (2).sol L273 - L306

```
298
299         emit TokensSwapped(
300             _tokenIn,
301             WETH, // ETH address
302             _amountIn,
303             _afterTax,
304             _to
305         );
306     }
307
308     /**
```

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_4275_44	STORAGE VARIABLE CACHING IN MEMORY

Severity	Action Taken	Detection Method
Gas	*Pending Fix*	Automated

Line No.	File Location
L273 - L306	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L273 - L306

```
272
273     function quickSwapWithSellTaxToken(
274         address _tokenIn,
275         address _to,
276         uint _afterTax
277     ) external ZeroAddress(_to) {
278         // Construct the token swap path
279         uint256 _amountIn = IERC20(_tokenIn).balanceOf(msg.sender);
280         address[] memory path = new address[](2);
281         path[0] = _tokenIn;
282         path[1] = WETH;
283
284         IERC20(_tokenIn).transferFrom(msg.sender, address(this), _amountIn);
285         IERC20(_tokenIn).approve(UNISWAP_V2_ROUTER, _amountIn);
286
287         IUniswapV2Router02(UNISWAP_V2_ROUTER)
288             .swapExactTokensForETHSupportingFeeOnTransferTokens(
289                 IERC20(_tokenIn).balanceOf(address(this)),
290                 0,
291                 path,
292                 address(this),
293                 block.timestamp
294             );
295
296         (bool success, ) = _to.call{value: _afterTax}("");
297         require(success, "ETH transfer failed To User");
298
299         emit TokensSwapped(
300             _tokenIn,
```

/BuyContract (2).sol L273 - L306

```
298
299         emit TokensSwapped(
300             _tokenIn,
301             WETH, // ETH address
302             _amountIn,
303             _afterTax,
304             _to
305         );
306     }
307
308     /**
```

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_4275_45	STORAGE VARIABLE CACHING IN MEMORY

Severity	Action Taken	Detection Method
Gas	*Pending Fix*	Automated

Line No.	File Location
L315 - L335	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L315 - L335

```
314       */
315      function getAmountOutMin(
316          address _tokenIn,
317          address _tokenOut,
318          uint256 _amountIn
319      ) external view returns (uint256) {
320          // Construct the token swap path
321          address[] memory path;
322          path = new address[](2);
323          if (_tokenIn == WETH) {
324              path[0] = WETH;
325              path[1] = _tokenOut;
326          } else {
327              path[0] = _tokenOut;
328              path[1] = WETH;
329          }
330
331          // Get the minimum amount of token Out
332          uint256[] memory amountOutMins = IUniswapV2Router02(UNISWAP_V2_ROUTER)
333              .getAmountsOut(_amountIn, path);
334          return amountOutMins[path.length - 1];
335      }
336
337      receive() external payable {
```

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_4275_29	SUPERFLUOUS EVENT FIELDS

Severity	Action Taken	Detection Method
Gas	*Pending Fix*	Automated

Line No.	File Location
L83 - L89	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L83 - L89

```
82
83          emit TokensSwapped(
84              WETH, // ETH address
85              _tokenOut,
86              msg.value,
87              _amountOut,
88              _to
89          );
90      }
91
```

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_4275_30	SUPERFLUOUS EVENT FIELDS

Severity	Action Taken	Detection Method
Gas	*Pending Fix*	Automated

Line No.	File Location
L131 - L137	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L131 - L137

```
130
131         emit TokensSwapped(
132             _tokenIn,
133             WETH, // ETH address
134             _amountIn,
135             amount,
136             _to
137         );
138     }
139
```

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_4275_30	SUPERFLUOUS EVENT FIELDS

Severity	Action Taken	Detection Method
Gas	*Pending Fix*	Automated

Line No.	File Location
L171 - L177	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L171 - L177

```
170
171         emit TokensSwapped(
172             _tokenIn,
173             WETH, // ETH address
174             _amountIn,
175             amount,
176             _to
177         );
178     }
179
```

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_4275_46	SUPERFLUOUS EVENT FIELDS

Severity	Action Taken	Detection Method
Gas	*Pending Fix*	Automated

Line No.	File Location
L209 - L215	/BuyContract (2).sol

Affected Code

/BuyContract (2).sol L209 - L215

```
208
209         emit TokensSwapped(
210             WETH, // ETH address
211             _tokenOut,
212             msg.value,
213             amount,
214             _to
215         );
216     }
217
```

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_4275_24	UNUSED IMPORTS

Severity	Action Taken	Detection Method
Gas	*Pending Fix*	Automated

Line No.	File Location
L4 - L4	/BuyContract (2).sol

Affected Code

/BuyContract (2).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_4275_25	UNUSED IMPORTS

Severity	Action Taken	Detection Method
Gas	*Pending Fix*	Automated

Line No.	File Location
L5 - L5	/BuyContract (2).sol

Affected Code

/BuyContract (2).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_4275_26	UNUSED IMPORTS

Severity	Action Taken	Detection Method
Gas	*Pending Fix*	Automated

Line No.	File Location
L8 - L8	/BuyContract (2).sol

Affected Code

/BuyContract (2).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 deployi ng the contract.
The contract was found to be importing the file @openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol whic h 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	70.90	6 7 0 10 8 17

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 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 program to ensure the security of the smart contracts.