SolidityScan

Powered by CRED SHIELDS

SELF PUBLISHED AUDIT REPORT
PUBLISHED ON SOLIDITYSCAN

Security Assessment

BYFCOIN

9 May 2024

This security assessment report was prepared by SolidityScan.com, a cloud-based Smart Contract Scanner.

Self-published

This automated audit report was Self-published by the user. To learn more about our published reports click here.

Table of Contents.

1. Vulnerability Classification and Severity

Description

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

● Critical

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

● High

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

● Medium

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

● Low

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

● Gas

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

● Informational

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

02. Executive Summary

BYFCOIN

Published on 09 May 2024

0xbB6f6F1A22b3A5E93EbdF2Ad001ED740B12695bC

https://etherscan.io/address/0xbB6f6F1A22b3A5E93EbdF2Ad0...

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

Security Score is GREAT

The SolidityScan 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 using SolidityScan 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 SolidityScan 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:

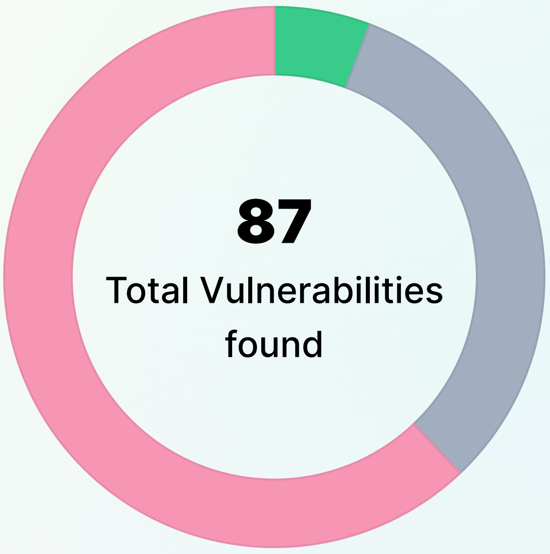
Various common and uncommon attack vectors will be investigated to ensure that the smart contracts are secure from malicious actors. The scanner modules find and flag issues related to Gas optimizations that help in reducing the overall Gas cost It scans and evaluates the codebase against industry best 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 SolidityScan Team recommends running regular audit scans to identify any vulnerabilities that are introduced after introduces new features or refactors the code.

3. **Findings** Summary

0xbB6f6F1A22b3A5E93EbdF2Ad001ED740B12695bC
ETHEREUM (Ethereum Mainnet) | View on Etherscan

Security Score	Scan duration	Lines of code
80.55/100	2 secs	273



0	0	0	5	28	54
Crit	High	Med	Low	Info	Gas

This audit report has not been verified by the SolidityScan team. To learn more about our published reports. click here.

ACTION TAKEN

0	2	0	86
✓ *Fixed*	*False Positive*	*Won't Fix*	⚠ *Pending Fix*

Bug ID	Severity	Bug Type	Detection Method	Line No	Status
SSB_318917_41	● Low	USE OF FLOATING PRAGMA	Automated	L2 - L2	⚠ *Pending Fix*
SSB_318917_42	● Low	LONG NUMBER LITERALS	Automated	L70 - L70	⚠ *Pending Fix*
SSB_318917_43	● Low	LONG NUMBER LITERALS	Automated	L74 - L74	⚠ *Pending Fix*
SSB_318917_44	● Low	LONG NUMBER LITERALS	Automated	L79 - L79	⚠ *Pending Fix*
SSB_318917_45	● Low	LONG NUMBER LITERALS	Automated	L84 - L84	⚠ *Pending Fix*
SSB_318917_34	● Low	MISSING EVENTS	Automated	L140 - L142	⚠ *Pending Fix*
SSB_318917_61	● Informational	BLOCK VALUES AS A PROXY FOR TIME	Automated	L136 - L136	⚠ *Pending Fix*
SSB_318917_62	● Informational	BLOCK VALUES AS A PROXY FOR TIME	Automated	L198 - L198	⚠ *Pending Fix*
SSB_318917_83	● Informational	IF-STATEMENT REFACTORING	Automated	L262 - L268	⚠ *Pending Fix*
SSB_318917_21	● Informational	MISSING UNDERSCORE IN NAMING VARIABLES	Automated	L46 - L46	⚠ *Pending Fix*
SSB_318917_22	● Informational	MISSING UNDERSCORE IN NAMING VARIABLES	Automated	L47 - L47	⚠ *Pending Fix*
SSB_318917_23	● Informational	MISSING UNDERSCORE IN NAMING VARIABLES	Automated	L48 - L48	⚠ *Pending Fix*
SSB_318917_24	● Informational	MISSING UNDERSCORE IN NAMING VARIABLES	Automated	L49 - L49	⚠ *Pending Fix*
SSB_318917_25	● Informational	MISSING UNDERSCORE IN NAMING VARIABLES	Automated	L5 - L9	⚠ *Pending Fix*
SSB_318917_26	● Informational	MISSING UNDERSCORE IN NAMING VARIABLES	Automated	L11 - L15	⚠ *Pending Fix*

Bug ID	Severity	Bug Type	Detection Method	Line No	Status
SSB_318917_27	● Informational	MISSING UNDERSCORE IN NAMING VARIABLES	Automated	L17 - L24	⚠ *Pending Fix*
SSB_318917_28	● Informational	MISSING UNDERSCORE IN NAMING VARIABLES	Automated	L26 - L30	⚠ *Pending Fix*
SSB_318917_11	● Informational	NAME MAPPING PARAMETERS	Automated	L46 - L46	⚠ *Pending Fix*
SSB_318917_12	● Informational	NAME MAPPING PARAMETERS	Automated	L47 - L47	⚠ *Pending Fix*
SSB_318917_13	● Informational	NAME MAPPING PARAMETERS	Automated	L48 - L48	⚠ *Pending Fix*
SSB_318917_14	● Informational	NAME MAPPING PARAMETERS	Automated	L49 - L49	⚠ *Pending Fix*
SSB_318917_75	● Informational	USE CALL INSTEAD OF TRANSFER OR SEND	Automated	L131 - L131	⚠ *Pending Fix*
SSB_318917_76	● Informational	USE CALL INSTEAD OF TRANSFER OR SEND	Automated	L173 - L173	⚠ *Pending Fix*
SSB_318917_77	● Informational	USE CALL INSTEAD OF TRANSFER OR SEND	Automated	L248 - L248	⚠ *Pending Fix*
SSB_318917_29	● Informational	USE SCIENTIFIC NOTATION	Automated	L70 - L70	⚠ *Pending Fix*
SSB_318917_30	● Informational	USE SCIENTIFIC NOTATION	Automated	L72 - L72	⚠ *Pending Fix*
SSB_318917_31	● Informational	USE SCIENTIFIC NOTATION	Automated	L79 - L79	⚠ *Pending Fix*
SSB_318917_32	● Informational	USE SCIENTIFIC NOTATION	Automated	L84 - L84	⚠ *Pending Fix*
SSB_318917_4	● Informational	VARIABLES SHOULD BE IMMUTABLE	Automated	L51 - L51	⚠ *Pending Fix*
SSB_318917_5	● Informational	VARIABLES SHOULD BE IMMUTABLE	Automated	L39 - L39	⚠ *Pending Fix*
SSB_318917_6	● Informational	VARIABLES SHOULD BE IMMUTABLE	Automated	L40 - L40	⚠ *Pending Fix*
SSB_318917_7	● Informational	VARIABLES SHOULD BE IMMUTABLE	Automated	L41 - L41	⚠ *Pending Fix*
SSB_318917_8	● Informational	VARIABLES SHOULD BE IMMUTABLE	Automated	L43 - L43	⚠ *Pending Fix*

Bug ID	Severity	Bug Type	Detection Method	Line No	Status
SSB_318917_9	● Informational	VARIABLES SHOULD BE IMMUTABLE	Automated	L52 - L52	⚠ *Pending Fix*
SSB_318917_2	● Gas	BYTES CONSTANT MORE EFFICIENT THAN STRING LITERAL	Automated	L36 - L36	⚠ *Pending Fix*
SSB_318917_3	● Gas	BYTES CONSTANT MORE EFFICIENT THAN STRING LITERAL	Automated	L37 - L37	⚠ *Pending Fix*
SSB_318917_84	● Gas	CHEAPER CONDITIONAL OPERATORS	Automated	L27 - L27	⚠ *Pending Fix*
SSB_318917_85	● Gas	CHEAPER CONDITIONAL OPERATORS	Automated	L151 - L151	⚠ *Pending Fix*
SSB_318917_86	● Gas	CHEAPER CONDITIONAL OPERATORS	Automated	L186 - L186	⚠ *Pending Fix*
SSB_318917_87	● Gas	CHEAPER CONDITIONAL OPERATORS	Automated	L195 - L195	⚠ *Pending Fix*
SSB_318917_88	● Gas	CHEAPER CONDITIONAL OPERATORS	Automated	L206 - L206	⚠ *Pending Fix*
SSB_318917_89	● Gas	CHEAPER CONDITIONAL OPERATORS	Automated	L234 - L234	⚠ *Pending Fix*
SSB_318917_90	● Gas	CHEAPER CONDITIONAL OPERATORS	Automated	L169 - L169	⚠ *Pending Fix*
SSB_318917_51	● Gas	CHEAPER INEQUALITIES IN IF()	Automated	L160 - L160	⚠ *Pending Fix*
SSB_318917_52	● Gas	CHEAPER INEQUALITIES IN IF()	Automated	L169 - L169	⚠ *Pending Fix*
SSB_318917_53	● Gas	CHEAPER INEQUALITIES IN REQUIRE()	Automated	L7 - L7	⚠ *Pending Fix*
SSB_318917_54	● Gas	CHEAPER INEQUALITIES IN REQUIRE()	Automated	L12 - L12	⚠ *Pending Fix*
SSB_318917_55	● Gas	CHEAPER INEQUALITIES IN REQUIRE()	Automated	L99 - L99	⚠ *Pending Fix*
SSB_318917_56	● Gas	CHEAPER INEQUALITIES IN REQUIRE()	Automated	L118 - L118	⚠ *Pending Fix*
SSB_318917_57	● Gas	CHEAPER INEQUALITIES IN REQUIRE()	Automated	L129 - L129	⚠ *Pending Fix*

Bug ID	Severity	Bug Type	Detection Method	Line No	Status
SSB_318917_58	● Gas	CHEAPER INEQUALITIES IN REQUIRE()	Automated	L152 - L152	⚠ *Pending Fix*
SSB_318917_59	● Gas	CHEAPER INEQUALITIES IN REQUIRE()	Automated	L216 - L216	⚠ *Pending Fix*
SSB_318917_60	● Gas	CHEAPER INEQUALITIES IN REQUIRE()	Automated	L235 - L235	⚠ *Pending Fix*
SSB_318917_10	● Gas	DEFINE CONSTRUCTOR AS PAYABLE	Automated	L68 - L85	⚠ *Pending Fix*
SSB_318917_19	● Gas	REVERTING FUNCTIONS CAN BE PAYABLE	Automated	L124 - L126	⚠ *Pending Fix*
SSB_318917_20	● Gas	REVERTING FUNCTIONS CAN BE PAYABLE	Automated	L128 - L133	⚠ *Pending Fix*
SSB_318917_63	● Gas	LONG REQUIRE/REVERT STRINGS	Automated	L22 - L22	⚠ *Pending Fix*
SSB_318917_64	● Gas	LONG REQUIRE/REVERT STRINGS	Automated	L99 - L99	⚠ *Pending Fix*
SSB_318917_65	● Gas	LONG REQUIRE/REVERT STRINGS	Automated	L151 - L151	⚠ *Pending Fix*
SSB_318917_66	● Gas	LONG REQUIRE/REVERT STRINGS	Automated	L195 - L195	⚠ *Pending Fix*
SSB_318917_67	● Gas	LONG REQUIRE/REVERT STRINGS	Automated	L206 - L206	⚠ *Pending Fix*
SSB_318917_68	● Gas	LONG REQUIRE/REVERT STRINGS	Automated	L234 - L234	⚠ *Pending Fix*
SSB_318917_15	● Gas	OPTIMIZING ADDRESS ID MAPPING	Automated	L46 - L46	⚠ *Pending Fix*
SSB_318917_16	● Gas	OPTIMIZING ADDRESS ID MAPPING	Automated	L47 - L47	⚠ *Pending Fix*
SSB_318917_17	● Gas	OPTIMIZING ADDRESS ID MAPPING	Automated	L48 - L48	⚠ *Pending Fix*
SSB_318917_18	● Gas	OPTIMIZING ADDRESS ID MAPPING	Automated	L49 - L49	⚠ *Pending Fix*
SSB_318917_72	● Gas	PUBLIC CONSTANTS CAN BE PRIVATE	Automated	L36 - L36	⚠ *Pending Fix*
SSB_318917_73	● Gas	PUBLIC CONSTANTS CAN BE PRIVATE	Automated	L37 - L37	⚠ *Pending Fix*

Bug ID	Severity	Bug Type	Detection Method	Line No	Status
SSB_318917_74	● Gas	PUBLIC CONSTANTS CAN BE PRIVATE	Automated	L38 - L38	⚠ *Pending Fix*
SSB_318917_1	● Gas	USE OF SAFEMATH LIBRARY	Automated	L34 - L34	⚠ *Pending Fix*
SSB_318917_35	● Gas	SMALLER DATA TYPES COST MORE	Automated	L70 - L70	⚠ *Pending Fix*
SSB_318917_36	● Gas	SMALLER DATA TYPES COST MORE	Automated	L72 - L72	⚠ *Pending Fix*
SSB_318917_37	● Gas	SMALLER DATA TYPES COST MORE	Automated	L79 - L79	⚠ *Pending Fix*
SSB_318917_38	● Gas	SMALLER DATA TYPES COST MORE	Automated	L84 - L84	⚠ *Pending Fix*
SSB_318917_39	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L68 - L85	⚠ *Pending Fix*
SSB_318917_39	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L68 - L85	⚠ *Pending Fix*
SSB_318917_46	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L96 - L102	⚠ *Pending Fix*
SSB_318917_47	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L110 - L114	⚠ *Pending Fix*
SSB_318917_48	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L116 - L122	⚠ *Pending Fix*
SSB_318917_49	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L128 - L133	⚠ *Pending Fix*
SSB_318917_50	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L145 - L182	⚠ *Pending Fix*
SSB_318917_50	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L145 - L182	⚠ *Pending Fix*
SSB_318917_50	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L145 - L182	⚠ *Pending Fix*
SSB_318917_50	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L145 - L182	⚠ *Pending Fix*
SSB_318917_78	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L185 - L190	⚠ *Pending Fix*
SSB_318917_79	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L193 - L203	⚠ *Pending Fix*

Bug ID	Severity	Bug Type	Detection Method	Line No	Status
SSB_318917_80	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L205 - L230	⚠ *Pending Fix*
SSB_318917_80	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L205 - L230	⚠ *Pending Fix*
SSB_318917_80	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L205 - L230	⚠ *Pending Fix*
SSB_318917_81	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L233 - L258	⚠ *Pending Fix*
SSB_318917_81	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L233 - L258	⚠ *Pending Fix*
SSB_318917_82	● Gas	STORAGE VARIABLE CACHING IN MEMORY	Automated	L261 - L272	⚠ *Pending Fix*
SSB_318917_33	● Gas	USE SELFBALANCE() INSTEAD OF ADDRESS(THIS).BALANCE	Automated	L129 - L129	⚠ *Pending Fix*
SSB_318917_70	● Gas	VARIABLES DECLARED BUT NEVER USED	Automated	L36 - L36	⚠ *Pending Fix*

4. Vulnerability Details

Bug ID	Bug Type	
SSB_318917_41	USE OF FLOATING PRAGMA	
Severity	**Action Taken**	**Detection Method**
● Low	*Pending Fix*	Automated

Line No.	File Location
L2 - L2	contract.sol

Affected Code

contract.sol L2 - L2

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.24;

library SafeMath {
```

Description

Solidity source files indicate the versions of the compiler they can be compiled with using a pragma directive at the t op 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:
`['contract.sol'] - ^0.8.24`

Remediation

It is recommended to use a fixed pragma version, as future compiler versions may handle certain language constructi ons 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 `^0.8.24` use `pragma solidity v0.8.24`, which is a stable and recommended version right now.

Bug ID	Bug Type
SSB_318917_42	LONG NUMBER LITERALS

Severity	Action Taken	Detection Method
● Low	*Pending Fix*	Automated

Line No.	File Location
L70 - L70	contract.sol

Affected Code

contract.sol L70 - L70

```
        owner = payable(msg.sender); // Set the owner to the address that deploys the contract
        totalSupply = 1000000000 * 10 ** uint256(decimals);
        maxSupply = totalSupply;
        maxWalletBalance = 20000 * 10 ** uint256(decimals);
```

Description

Solidity supports multiple rational and integer literals, including decimal fractions and scientific notations. The use of very large numbers with too many digits was detected in the code that could have been optimized using a different n otation also supported by Solidity.
The value 1000000000 was detected on line 70.

Remediation

Scientific notation in the form of `2e10` is also supported, where the mantissa can be fractional but the exponent has to be an integer. The literal `MeE` is equivalent to `M * 10**E`. Examples include `2e10`, `2e10`, `2e-10`, `2.5e1`, as suggested in official solidity documentation `https://docs.soliditylang.org/en/latest/types.html#rational-and-integer-literals`

Bug ID

SSB_318917_43

Bug Type

LONG NUMBER LITERALS

Severity

● Low

Action Taken

Pending Fix

Detection Method

Automated

Line No.

L74 - L74

File Location

contract.sol

Affected Code

contract.sol L74 - L74

```
        lockTimeBlocks = 105120000; // Equivalent to approximately 2 years with 15 seconds per block
        rate = 100000; // Initial rate: 100000 BYF per 1 ETH

        balances[msg.sender] = totalSupply;
```

Description

Solidity supports multiple rational and integer literals, including decimal fractions and scientific notations. The use of very large numbers with too many digits was detected in the code that could have been optimized using a different n otation also supported by Solidity.
The value 100000 was detected on line 74.

Remediation

Scientific notation in the form of `2e10` is also supported, where the mantissa can be fractional but the exponent has to be an integer. The literal `MeE` is equivalent to `M * 10**E`. Examples include `2e10`, `2e10`, `2e-10`, `2.5e1`, as suggested in official solidity documentation `https://docs.soliditylang.org/en/latest/types.html#rational-and-integer-literals`

Bug ID	Bug Type	
SSB_318917_44	**LONG NUMBER LITERALS**	
Severity	Action Taken	Detection Method
● Low	*Pending Fix*	Automated

Line No.	File Location
L79 - L79	contract.sol

Affected Code

contract.sol L79 - L79

```
        // Lock a portion of the owner's wallet balance for 2 years
        uint256 lockedBalance = 100000000 * 10 ** uint256(decimals);
        _lockTokens(msg.sender, lockedBalance, lockTimeBlocks);
```

Description

Solidity supports multiple rational and integer literals, including decimal fractions and scientific notations. The use of very large numbers with too many digits was detected in the code that could have been optimized using a different n otation also supported by Solidity.
The value 100000000 was detected on line 79.

Remediation

Scientific notation in the form of `2e10` is also supported, where the mantissa can be fractional but the exponent has to be an integer. The literal `MeE` is equivalent to `M * 10**E`. Examples include `2e10`, `2e10`, `2e-10`, `2.5e1`, as suggested in official solidity documentation `https://docs.soliditylang.org/en/latest/types.html#rational-and-integer-literals`

Bug ID	Bug Type	
SSB_318917_45	LONG NUMBER LITERALS	
Severity	**Action Taken**	**Detection Method**
● Low	*Pending Fix*	Automated

Line No.	File Location
L84 - L84	contract.sol

Affected Code

contract.sol L84 - L84

```
        tradingAddress = address(this);
        balances[tradingAddress] = 100000000 * 10 ** uint256(decimals);
    }

```

Description

Solidity supports multiple rational and integer literals, including decimal fractions and scientific notations. The use of very large numbers with too many digits was detected in the code that could have been optimized using a different n otation also supported by Solidity.
The value 100000000 was detected on line 84.

Remediation

Scientific notation in the form of `2e10` is also supported, where the mantissa can be fractional but the exponent has to be an integer. The literal `MeE` is equivalent to `M * 10**E`. Examples include `2e10`, `2e10`, `2e-10`, `2.5e1`, as suggested in official solidity documentation `https://docs.soliditylang.org/en/latest/types.html#rational-and-integer-literals`

Bug ID	Bug Type	
SSB_318917_34	**MISSING EVENTS**	
Severity	Action Taken	Detection Method
● Low	*Pending Fix*	Automated

Line No.	File Location
L140 - L142	contract.sol

Affected Code

contract.sol L140 - L142

```
    // Fallback function to receive Ether
    receive() external payable {
        emit Received(msg.sender, msg.value);
    }

    // Internal transfer function
```

Description

Events are inheritable members of contracts. When you call them, they cause the arguments to be stored in the trans action'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 ke ep track of the transactions.
The contract BYFCOIN was found to be missing these events on the function which would make it difficult or impossi ble 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

SSB_318917_61

Bug Type

BLOCK VALUES AS A PROXY FOR TIME

Severity

● Informational

Action Taken

Pending Fix

Detection Method

Automated

Line No.

L136 - L136

File Location

contract.sol

Affected Code

contract.sol L136 - L136

```
    function isUnlocked(address account) external view returns (bool) {
        return unlockTime[account] <= block.timestamp;
    }

```

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

SSB_318917_62

Bug Type

BLOCK VALUES AS A PROXY FOR TIME

Severity

● Informational

Action Taken

Pending Fix

Detection Method

Automated

Line No.

L198 - L198

File Location

contract.sol

Affected Code

contract.sol L198 - L198

```
        // Calculate the unlock timestamp based on the current block timestamp and the lock duration
        uint256 unlockTimestamp = block.timestamp + lockDuration;

        unlockTime[account] = unlockTimestamp;
```

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

SSB_318917_83

Bug Type

IF-STATEMENT REFACTORING

Severity

● Informational

Action Taken

Pending Fix

Detection Method

Automated

Line No.

L262 - L268

File Location

contract.sol

Affected Code

contract.sol L262 - L268

```
    function _updateRate(bool isBuy) private {
        if (isBuy) {
            // Decrease rate by 0.4% after each buy
            rate = rate.mul(996).div(1000);
        } else {
            // Increase rate by 0.1% after each sell
            rate = rate.mul(1001).div(1000);
        }

        // Emit the RateUpdated event with the new rate
```

Description

In Solidity, we aim to write clear, efficient code that is both easy to understand and maintain. If statements can be co nverted to ternary operators. While using ternary operators instead of if/else statements can sometimes lead to more concise code, it's crucial to understand the trade-offs involved.

Remediation

To optimize your Solidity code, consider converting simple if/else statements to ternary operators, particularly for sin gle-line arithmetic or logical operations. Utilizing ternary operators can improve code conciseness and readability. Ho wever, be mindful of code complexity and readability concerns. If the if/else statement is not single-line or involves m ultiple operations, retaining it for clarity is advisable.

Bug ID	Bug Type
SSB_318917_21	MISSING UNDERSCORE IN NAMING VARIABLES

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

Line No.	File Location
L46 - L46	contract.sol

Affected Code

contract.sol L46 - L46

```
    mapping(address => uint256) private balances;
    mapping(address => mapping(address => uint256)) private allowances;
    mapping(address => uint256) private unlockTime;
```

Description

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

Remediation

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

Bug ID

SSB_318917_22

Bug Type

MISSING UNDERSCORE IN NAMING VARIABLES

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

Line No.	File Location
L47 - L47	contract.sol

Affected Code

contract.sol L47 - L47

```
    mapping(address => uint256) private balances;
    mapping(address => mapping(address => uint256)) private allowances;
    mapping(address => uint256) private unlockTime;
    mapping(address => bool) private mutex; // Mutex lock
```

Description

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

Remediation

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

Bug ID

SSB_318917_23

Bug Type

MISSING UNDERSCORE IN NAMING VARIABLES

Severity

● Informational

Action Taken

Pending Fix

Detection Method

Automated

Line No.

L48 - L48

File Location

contract.sol

Affected Code

contract.sol L48 - L48

```
    mapping(address => mapping(address => uint256)) private allowances;
    mapping(address => uint256) private unlockTime;
    mapping(address => bool) private mutex; // Mutex lock
```

Description

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

Remediation

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

Bug ID

SSB_318917_24

Bug Type

MISSING UNDERSCORE IN NAMING VARIABLES

Severity

● Informational

Action Taken

Pending Fix

Detection Method

Automated

Line No.

L49 - L49

File Location

contract.sol

Affected Code

contract.sol L49 - L49

```
    mapping(address => uint256) private unlockTime;
    mapping(address => bool) private mutex; // Mutex lock

    address payable public owner;
```

Description

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

Remediation

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

Bug ID

SSB_318917_25

Bug Type

MISSING UNDERSCORE IN NAMING VARIABLES

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

Line No.	File Location
L5 - L9	contract.sol

Affected Code

contract.sol L5 - L9

```
library SafeMath {
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, 'SafeMath: addition overflow');
        return c;
    }

    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
```

Description

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

Remediation

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

Bug ID	Bug Type	
SSB_318917_26	**MISSING UNDERSCORE IN NAMING VARIABLES**	
Severity	Action Taken	Detection Method
● Informational	*Pending Fix*	Automated

Line No.	File Location
L11 - L15	contract.sol

Affected Code

contract.sol L11 - L15

```

    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a, 'SafeMath: subtraction overflow');
        uint256 c = a - b;
        return c;
    }

    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
```

Description

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

Remediation

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

Bug ID

SSB_318917_27

Bug Type

MISSING UNDERSCORE IN NAMING VARIABLES

Severity

● Informational

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L17 - L24

File Location

contract.sol

Affected Code

contract.sol L17 - L24

```
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) {
            return 0;
        }
        uint256 c = a * b;
        require(c / a == b, 'SafeMath: multiplication overflow');
        return c;
    }

    function div(uint256 a, uint256 b) internal pure returns (uint256) {
```

Description

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

Remediation

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

Bug ID

SSB_318917_28

Bug Type

MISSING UNDERSCORE IN NAMING VARIABLES

Severity

● Informational

Action Taken

Pending Fix

Detection Method

Automated

Line No.

L26 - L30

File Location

contract.sol

Affected Code

contract.sol L26 - L30

```
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, 'SafeMath: division by zero');
        uint256 c = a / b;
        return c;
    }
}
```

Description

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

Remediation

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

Bug ID	Bug Type	
SSB_318917_11	NAME MAPPING PARAMETERS	
Severity	Action Taken	Detection Method
● Informational	*Pending Fix*	Automated

Line No.	File Location
L46 - L46	contract.sol

Affected Code

contract.sol L46 - L46

```
    mapping(address => uint256) private balances;
    mapping(address => mapping(address => uint256)) private allowances;
    mapping(address => uint256) private unlockTime;
```

Description

After Solidity 0.8.18, a feature was introduced to name mapping parameters. This helps in defining a purpose for each mapping and makes the code more descriptive.

Remediation

It is recommended to name the mapping parameters if Solidity 0.8.18 and above is used.

Bug ID

SSB_318917_12

Bug Type

NAME MAPPING PARAMETERS

Severity

● Informational

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L47 - L47

File Location

contract.sol

Affected Code

contract.sol L47 - L47

```
    mapping(address => uint256) private balances;
    mapping(address => mapping(address => uint256)) private allowances;
    mapping(address => uint256) private unlockTime;
    mapping(address => bool) private mutex; // Mutex lock
```

Description

After Solidity 0.8.18, a feature was introduced to name mapping parameters. This helps in defining a purpose for each mapping and makes the code more descriptive.

Remediation

It is recommended to name the mapping parameters if Solidity 0.8.18 and above is used.

Bug ID	Bug Type	
SSB_318917_13	**NAME MAPPING PARAMETERS**	
Severity	Action Taken	Detection Method
● Informational	⚠ *Pending Fix*	Automated

Line No.	File Location
L48 - L48	contract.sol

Affected Code

contract.sol L48 - L48

```
mapping(address => mapping(address => uint256)) private allowances;
mapping(address => uint256) private unlockTime;
mapping(address => bool) private mutex; // Mutex lock
```

Description

After Solidity 0.8.18, a feature was introduced to name mapping parameters. This helps in defining a purpose for each mapping and makes the code more descriptive.

Remediation

It is recommended to name the mapping parameters if Solidity 0.8.18 and above is used.

Bug ID

SSB_318917_14

Bug Type

NAME MAPPING PARAMETERS

Severity

● Informational

Action Taken

Pending Fix

Detection Method

Automated

Line No.

L49 - L49

File Location

contract.sol

Affected Code

contract.sol L49 - L49

```
    mapping(address => uint256) private unlockTime;
    mapping(address => bool) private mutex; // Mutex lock

    address payable public owner;
```

Description

After Solidity 0.8.18, a feature was introduced to name mapping parameters. This helps in defining a purpose for each mapping and makes the code more descriptive.

Remediation

It is recommended to name the mapping parameters if Solidity 0.8.18 and above is used.

Bug ID	Bug Type	
SSB_318917_75	**USE CALL INSTEAD OF TRANSFER OR SEND**	
Severity	Action Taken	Detection Method
● Informational	⚠ *Pending Fix*	Automated

Line No.	File Location
L131 - L131	contract.sol

Affected Code

contract.sol L131 - L131

```
            owner.transfer(amount); // Transfer the specified amount to the owner
            emit Withdraw(owner, amount); // Emit withdrawal event
        }
```

Description

The contract was found to be using `transfer` or `send` function call. This is unsafe as `transfer` has hard coded gas 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	
SSB_318917_76	**USE CALL INSTEAD OF TRANSFER OR SEND**	
Severity	Action Taken	Detection Method
● Informational	⚠ *Pending Fix*	Automated

Line No.	File Location
L173 - L173	contract.sol

Affected Code

contract.sol L173 - L173

```
            // Transfer ETH tax to owner's wallet
            owner.transfer(ethTaxAmount);
            emit Transfer(from, owner, taxAmount);
            emit TaxDeducted(from, owner, ethTaxAmount); // Emit tax deduction event
```

Description

The contract was found to be using `transfer` or `send` function call. This is unsafe as `transfer` has hard coded gas 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

SSB_318917_77

Bug Type

USE CALL INSTEAD OF TRANSFER OR SEND

Severity

● Informational

Action Taken

Pending Fix

Detection Method

Automated

Line No.

L248 - L248

File Location

contract.sol

Affected Code

contract.sol L248 - L248

```
        // Transfer ETH to the seller
        payable(msg.sender).transfer(ethAmount);

        // Emit the Sold event
```

Description

The contract was found to be using `transfer` or `send` function call. This is unsafe as `transfer` has hard coded gas 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	
SSB_318917_29	**USE SCIENTIFIC NOTATION**	
Severity	Action Taken	Detection Method
● Informational	⚠ *Pending Fix*	Automated

Line No.	File Location
L70 - L70	contract.sol

Affected Code

contract.sol L70 - L70

```
        owner = payable(msg.sender); // Set the owner to the address that deploys the contract
        totalSupply = 1000000000 * 10 ** uint256(decimals);
        maxSupply = totalSupply;
        maxWalletBalance = 20000 * 10 ** uint256(decimals);
```

Description

Although the Solidity compiler can optimize exponentiation, it is recommended to prioritize idioms not reliant on com piler optimization. Utilizing scientific notation enhances code clarity, making it more self-explanatory and aligning wit h best practices in Solidity development.

Remediation

Enhance code readability by replacing exponentiation with scientific notation where applicable. This practice not onl y aligns with best practices but also reduces the reliance on compiler optimization, contributing to more robust and h uman-friendly Solidity code.

Bug ID

SSB_318917_30

Bug Type

USE SCIENTIFIC NOTATION

Severity

● Informational

Action Taken

Pending Fix

Detection Method

Automated

Line No.

L72 - L72

File Location

contract.sol

Affected Code

contract.sol L72 - L72

```
        maxSupply = totalSupply;
        maxWalletBalance = 20000 * 10 ** uint256(decimals);
        lockTimeBlocks = 105120000; // Equivalent to approximately 2 years with 15 seconds per block
        rate = 100000; // Initial rate: 100000 BYF per 1 ETH
```

Description

Although the Solidity compiler can optimize exponentiation, it is recommended to prioritize idioms not reliant on com piler optimization. Utilizing scientific notation enhances code clarity, making it more self-explanatory and aligning wit h best practices in Solidity development.

Remediation

Enhance code readability by replacing exponentiation with scientific notation where applicable. This practice not onl y aligns with best practices but also reduces the reliance on compiler optimization, contributing to more robust and h uman-friendly Solidity code.

Bug ID	Bug Type	
SSB_318917_31	**USE SCIENTIFIC NOTATION**	
Severity	Action Taken	Detection Method
● Informational	⚠ *Pending Fix*	Automated

Line No.	File Location
L79 - L79	contract.sol

Affected Code

contract.sol L79 - L79

```
        // Lock a portion of the owner's wallet balance for 2 years
        uint256 lockedBalance = 100000000 * 10 ** uint256(decimals);
        _lockTokens(msg.sender, lockedBalance, lockTimeBlocks);
```

Description

Although the Solidity compiler can optimize exponentiation, it is recommended to prioritize idioms not reliant on com piler optimization. Utilizing scientific notation enhances code clarity, making it more self-explanatory and aligning wit h best practices in Solidity development.

Remediation

Enhance code readability by replacing exponentiation with scientific notation where applicable. This practice not onl y aligns with best practices but also reduces the reliance on compiler optimization, contributing to more robust and h uman-friendly Solidity code.

Bug ID

SSB_318917_32

Bug Type

USE SCIENTIFIC NOTATION

Severity

● Informational

Action Taken

Pending Fix

Detection Method

Automated

Line No.

L84 - L84

File Location

contract.sol

Affected Code

contract.sol L84 - L84

```
        tradingAddress = address(this);
        balances[tradingAddress] = 100000000 * 10 ** uint256(decimals);
    }

```

Description

Although the Solidity compiler can optimize exponentiation, it is recommended to prioritize idioms not reliant on com piler optimization. Utilizing scientific notation enhances code clarity, making it more self-explanatory and aligning wit h best practices in Solidity development.

Remediation

Enhance code readability by replacing exponentiation with scientific notation where applicable. This practice not onl y aligns with best practices but also reduces the reliance on compiler optimization, contributing to more robust and h uman-friendly Solidity code.

Bug ID

SSB_318917_4

Bug Type

VARIABLES SHOULD BE IMMUTABLE

Severity

● Informational

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L51 - L51

File Location

contract.sol

Affected Code

contract.sol L51 - L51

```
    address payable public owner;
    address public tradingAddress;
```

Description

Constants and Immutables should be used in their appropriate contexts.
`constant` should only be used for literal values written into the code. `immutable` variables should be used for expressions, or values calculated in, or passed into the constructor.

Remediation

It is recommended to use `immutable` instead of `constant`.

Bug ID

SSB_318917_5

Bug Type

VARIABLES SHOULD BE IMMUTABLE

Severity

● Informational

Action Taken

Pending Fix

Detection Method

Automated

Line No.

L39 - L39

File Location

contract.sol

Affected Code

contract.sol L39 - L39

```
        uint8 public constant decimals = 18;
        uint256 public totalSupply;
        uint256 public maxSupply;
        uint256 public maxWalletBalance;
```

Description

Constants and Immutables should be used in their appropriate contexts.
`constant` should only be used for literal values written into the code. `immutable` variables should be used for ex pressions, or values calculated in, or passed into the constructor.

Remediation

It is recommended to use `immutable` instead of `constant`.

Bug ID	Bug Type	
SSB_318917_6	VARIABLES SHOULD BE IMMUTABLE	
Severity	**Action Taken**	**Detection Method**
● Informational	*Pending Fix*	Automated

Line No.	File Location
L40 - L40	contract.sol

Affected Code

contract.sol L40 - L40

```
    uint256 public totalSupply;
    uint256 public maxSupply;
    uint256 public maxWalletBalance;
    uint256 public taxRate = 3; // 3% tax rate represented as a decimal fraction
```

Description

Constants and Immutables should be used in their appropriate contexts.
`constant` should only be used for literal values written into the code. `immutable` variables should be used for ex pressions, or values calculated in, or passed into the constructor.

Remediation

It is recommended to use `immutable` instead of `constant`.

Bug ID

SSB_318917_7

Bug Type

VARIABLES SHOULD BE IMMUTABLE

Severity

● Informational

Action Taken

Pending Fix

Detection Method

Automated

Line No.

L41 - L41

File Location

contract.sol

Affected Code

contract.sol L41 - L41

```
    uint256 public maxSupply;
    uint256 public maxWalletBalance;
    uint256 public taxRate = 3; // 3% tax rate represented as a decimal fraction
    uint256 public lockTimeBlocks; // Lock duration in blocks
```

Description

Constants and Immutables should be used in their appropriate contexts.
`constant` should only be used for literal values written into the code. `immutable` variables should be used for ex pressions, or values calculated in, or passed into the constructor.

Remediation

It is recommended to use `immutable` instead of `constant`.

Bug ID

SSB_318917_8

Bug Type

VARIABLES SHOULD BE IMMUTABLE

Severity

● Informational

Action Taken

Pending Fix

Detection Method

Automated

Line No.

L43 - L43

File Location

contract.sol

Affected Code

contract.sol L43 - L43

```
        uint256 public taxRate = 3; // 3% tax rate represented as a decimal fraction
        uint256 public lockTimeBlocks; // Lock duration in blocks
        uint256 public rate; // Rate of swap (BYF per ETH)
```

Description

Constants and Immutables should be used in their appropriate contexts.
`constant` should only be used for literal values written into the code. `immutable` variables should be used for ex pressions, or values calculated in, or passed into the constructor.

Remediation

It is recommended to use `immutable` instead of `constant`.

Bug ID	Bug Type	
SSB_318917_9	VARIABLES SHOULD BE IMMUTABLE	
Severity	**Action Taken**	**Detection Method**
● Informational	*Pending Fix*	Automated

Line No.	File Location
L52 - L52	contract.sol

Affected Code

contract.sol L52 - L52

```
    address payable public owner;
    address public tradingAddress;

    event Transfer(address indexed from, address indexed to, uint256 value);
```

Description

Constants and Immutables should be used in their appropriate contexts.
`constant` should only be used for literal values written into the code. `immutable` variables should be used for ex pressions, or values calculated in, or passed into the constructor.

Remediation

It is recommended to use `immutable` instead of `constant`.

Bug ID	Bug Type	
SSB_318917_2	BYTES CONSTANT MORE EFFICIENT THAN STRING LITERAL	
Severity	**Action Taken**	**Detection Method**
● Gas	⚠ *Pending Fix*	Automated

Line No.	File Location
L36 - L36	contract.sol

Affected Code

contract.sol L36 - L36

```
    string public constant name = "BYFCOIN";
    string public constant symbol = "BYF";
    uint8 public constant decimals = 18;
```

Description

The contract was found to be using name string constant. This can be optimized by using `bytes32 constant` to s ave gas.

Remediation

Unless explicitly required, if the string is lesser than 32 bytes, it is recommended to use `bytes32 constant` instea d of a `string constant` as it'll save some gas.

Bug ID

SSB_318917_3

Bug Type

BYTES CONSTANT MORE EFFICIENT THAN STRING LITERAL

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

Line No.	File Location
L37 - L37	contract.sol

Affected Code

contract.sol L37 - L37

```
        string public constant name = "BYFCOIN";
        string public constant symbol = "BYF";
        uint8 public constant decimals = 18;
        uint256 public totalSupply;
```

Description

The contract was found to be using symbol string constant. This can be optimized by using `bytes32 constant` to save gas.

Remediation

Unless explicitly required, if the string is lesser than 32 bytes, it is recommended to use `bytes32 constant` instead of a `string constant` as it'll save some gas.

Bug ID

SSB_318917_84

Bug Type

CHEAPER CONDITIONAL OPERATORS

Severity

● Gas

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L27 - L27

File Location

contract.sol

Affected Code

contract.sol L27 - L27

```
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, 'SafeMath: division by zero');
        uint256 c = a / b;
        return c;
```

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

SSB_318917_85

Bug Type

CHEAPER CONDITIONAL OPERATORS

Severity

● Gas

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L151 - L151

File Location

contract.sol

Affected Code

contract.sol L151 - L151

```
        require(to != address(0), "Invalid address");
        require(value > 0, "Transfer value must be greater than zero");
        require(balances[from] >= value, "Insufficient balance");
```

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	
SSB_318917_86	CHEAPER CONDITIONAL OPERATORS	
Severity	**Action Taken**	**Detection Method**
● Gas	*Pending Fix*	Automated

Line No.	File Location
L186 - L186	contract.sol

Affected Code

contract.sol L186 - L186

```
    function _calculateEthAmount(uint256 byfAmount) private view returns (uint256) {
        require(rate > 0, "Rate must be greater than zero");
        // Calculate ETH amount based on current rate
        uint256 ethAmount = byfAmount.div(rate);
```

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

SSB_318917_87

Bug Type

CHEAPER CONDITIONAL OPERATORS

Severity

● Gas

Action Taken

Pending Fix

Detection Method

Automated

Line No.

L195 - L195

File Location

contract.sol

Affected Code

contract.sol L195 - L195

```
        require(account != address(0), "Invalid address");
        require(lockDuration > 0, "Lock duration must be greater than zero");

        // Calculate the unlock timestamp based on the current block timestamp and the lock duration
```

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

SSB_318917_88

Bug Type

CHEAPER CONDITIONAL OPERATORS

Severity

● Gas

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L206 - L206

File Location

contract.sol

Affected Code

contract.sol L206 - L206

```
    function buyBYF(uint256 ethAmountInWei) external payable {
        require(ethAmountInWei > 0, "ETH amount must be greater than zero");

        // Implement mutex lock at the beginning of the function
```

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

SSB_318917_89

Bug Type

CHEAPER CONDITIONAL OPERATORS

Severity

● Gas

Action Taken

Pending Fix

Detection Method

Automated

Line No.

L234 - L234

File Location

contract.sol

Affected Code

contract.sol L234 - L234

```
    function sellBYF(uint256 byfAmount) external {
        require(byfAmount > 0, "BYF amount must be greater than zero");
        require(balances[msg.sender] >= byfAmount, "Insufficient BYF balance");

```

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

SSB_318917_90

Bug Type

CHEAPER CONDITIONAL OPERATORS

Severity

● Gas

Action Taken

Pending Fix

Detection Method

Automated

Line No.

L169 - L169

File Location

contract.sol

Affected Code

contract.sol L169 - L169

```
        if (taxAmount > 0) {
            // Convert tax amount to ETH
            uint256 ethTaxAmount = _calculateEthAmount(taxAmount);
```

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

SSB_318917_51

Bug Type

CHEAPER INEQUALITIES IN IF()

Severity

● Gas

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L160 - L160

File Location

contract.sol

Affected Code

contract.sol L160 - L160

```
        if (from != owner && to != owner && balances[to].add(transferAmount) > maxWalletBalance) {
            uint256 excessTokens = balances[to].add(transferAmount).sub(maxWalletBalance);
            _lockTokens(to, excessTokens, lockTimeBlocks);
```

Description

The contract was found to be doing comparisons using inequalities inside the if statement.
When inside the `if` statements, non-strict inequalities `(>=, <=)` are usually cheaper than the strict equalities `(>, <).`

Remediation

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

Bug ID

SSB_318917_52

Bug Type

CHEAPER INEQUALITIES IN IF()

Severity

● Gas

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L169 - L169

File Location

contract.sol

Affected Code

contract.sol L169 - L169

```
        if (taxAmount > 0) {
            // Convert tax amount to ETH
            uint256 ethTaxAmount = _calculateEthAmount(taxAmount);
```

Description

The contract was found to be doing comparisons using inequalities inside the if statement.
When inside the `if` statements, non-strict inequalities `(>=, <=)` are usually cheaper than the strict equalities `(>, <).`

Remediation

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

Bug ID	Bug Type	
SSB_318917_53	CHEAPER INEQUALITIES IN REQUIRE()	
Severity	**Action Taken**	**Detection Method**
● Gas	⚠ *Pending Fix*	Automated

Line No.	File Location
L7 - L7	contract.sol

Affected Code

contract.sol L7 - L7

```
        uint256 c = a + b;
        require(c >= a, 'SafeMath: addition overflow');
        return c;
    }
```

Description

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

Remediation

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

Bug ID

SSB_318917_54

Bug Type

CHEAPER INEQUALITIES IN REQUIRE()

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

Line No.	File Location
L12 - L12	contract.sol

Affected Code

contract.sol L12 - L12

```
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a, 'SafeMath: subtraction overflow');
        uint256 c = a - b;
        return c;
```

Description

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

Remediation

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

Bug ID

SSB_318917_55

Bug Type

CHEAPER INEQUALITIES IN REQUIRE()

Severity

● Gas

Action Taken

Pending Fix

Detection Method

Automated

Line No.

L99 - L99

File Location

contract.sol

Affected Code

contract.sol L99 - L99

```
        uint256 currentAllowance = allowances[from][msg.sender];
        require(currentAllowance >= value, "Transfer amount exceeds allowance");
         allowances[from][msg.sender] = currentAllowance.sub(value);
         return true;
```

Description

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

Remediation

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

Bug ID

SSB_318917_56

Bug Type

CHEAPER INEQUALITIES IN REQUIRE()

Severity

● Gas

Action Taken

Pending Fix

Detection Method

Automated

Line No.

L118 - L118

File Location

contract.sol

Affected Code

contract.sol L118 - L118

```
        uint256 currentAllowance = allowances[msg.sender][spender];
        require(currentAllowance >= subtractedValue, "Decreased allowance below zero");
        allowances[msg.sender][spender] = currentAllowance.sub(subtractedValue);
        emit Approval(msg.sender, spender, allowances[msg.sender][spender]);
```

Description

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

Remediation

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

Bug ID	Bug Type	
SSB_318917_57	**CHEAPER INEQUALITIES IN REQUIRE()**	
Severity	Action Taken	Detection Method
● Gas	*Pending Fix*	Automated

Line No.	File Location
L129 - L129	contract.sol

Affected Code

contract.sol L129 - L129

```
    function withdrawEther(uint256 amount) external onlyOwner {
        require(amount <= address(this).balance, "Insufficient contract balance");

        owner.transfer(amount); // Transfer the specified amount to the owner
```

Description

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

Remediation

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

Bug ID

SSB_318917_58

Bug Type

CHEAPER INEQUALITIES IN REQUIRE()

Severity

● Gas

Action Taken

Pending Fix

Detection Method

Automated

Line No.

L152 - L152

File Location

contract.sol

Affected Code

contract.sol L152 - L152

```
        require(value > 0, "Transfer value must be greater than zero");
        require(balances[from] >= value, "Insufficient balance");

        // Calculate the tax amount based on the tax rate
```

Description

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

Remediation

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

Bug ID

SSB_318917_59

Bug Type

CHEAPER INEQUALITIES IN REQUIRE()

Severity

● Gas

Action Taken

Pending Fix

Detection Method

Automated

Line No.

L216 - L216

File Location

contract.sol

Affected Code

contract.sol L216 - L216

```
        // Ensure that the contract has enough BYF tokens to fulfill the purchase
        require(balances[tradingAddress] >= byfAmount, "Insufficient BYF balance");

        // Transfer BYF tokens to the buyer
```

Description

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

Remediation

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

Bug ID	Bug Type	
SSB_318917_60	**CHEAPER INEQUALITIES IN REQUIRE()**	
Severity	Action Taken	Detection Method
● Gas	⚠ *Pending Fix*	Automated

Line No.	File Location
L235 - L235	contract.sol

Affected Code

contract.sol L235 - L235

```
        require(byfAmount > 0, "BYF amount must be greater than zero");
        require(balances[msg.sender] >= byfAmount, "Insufficient BYF balance");

        // Implement mutex lock at the beginning of the function
```

Description

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

Remediation

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

Bug ID

SSB_318917_10

Bug Type

DEFINE CONSTRUCTOR AS PAYABLE

Severity

● Gas

Action Taken

Pending Fix

Detection Method

Automated

Line No.

L68 - L85

File Location

contract.sol

Affected Code

contract.sol L68 - L85

```
    constructor() {
        owner = payable(msg.sender); // Set the owner to the address that deploys the contract
        totalSupply = 1000000000 * 10 ** uint256(decimals);
        maxSupply = totalSupply;
        maxWalletBalance = 20000 * 10 ** uint256(decimals);
        lockTimeBlocks = 105120000; // Equivalent to approximately 2 years with 15 seconds per block
        rate = 100000; // Initial rate: 100000 BYF per 1 ETH

        balances[msg.sender] = totalSupply;

        // Lock a portion of the owner's wallet balance for 2 years
        uint256 lockedBalance = 100000000 * 10 ** uint256(decimals);
        _lockTokens(msg.sender, lockedBalance, lockTimeBlocks);

        // Allocate 100,000,000 BYF for trading
        tradingAddress = address(this);
        balances[tradingAddress] = 100000000 * 10 ** uint256(decimals);
    }

    function balanceOf(address account) external view returns (uint256) {
        return balances[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 deployme nt.

Remediation

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

Bug ID

SSB_318917_19

Bug Type

REVERTING FUNCTIONS CAN BE PAYABLE

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

Line No.	File Location
L124 - L126	contract.sol

Affected Code

contract.sol — L124 - L126

```
    function withdrawTokens(uint256 amount) external onlyOwner {
        _transfer(tradingAddress, msg.sender, amount);
    }

    function withdrawEther(uint256 amount) external onlyOwner {
```

Description

If a function modifier such as `onlyOwner` is used, the function will revert if a normal user tries to pay the function. Marking the function as payable will lower the gas cost for legitimate callers because the compiler will not include ch ecks for whether a payment was provided.

Remediation

In the above code, the `onlyOwner` modifier ensures that only the contract owner can execute the `withdrawTokens`. If a normal user attempts to call this function, the transaction will automatically revert. By marking the `withdrawTokens` as payable, we can optimize gas costs for legitimate callers since the compiler will skip the checks for paym ent.

Bug ID

SSB_318917_20

Bug Type

REVERTING FUNCTIONS CAN BE PAYABLE

Severity

● Gas

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L128 - L133

File Location

contract.sol

Affected Code

contract.sol L128 - L133

```

    function withdrawEther(uint256 amount) external onlyOwner {
        require(amount <= address(this).balance, "Insufficient contract balance");

        owner.transfer(amount); // Transfer the specified amount to the owner
        emit Withdraw(owner, amount); // Emit withdrawal event
    }

    function isUnlocked(address account) external view returns (bool) {
```

Description

If a function modifier such as `onlyOwner` is used, the function will revert if a normal user tries to pay the function. Marking the function as payable will lower the gas cost for legitimate callers because the compiler will not include ch ecks for whether a payment was provided.

Remediation

In the above code, the `onlyOwner` modifier ensures that only the contract owner can execute the `withdrawEthe r`. If a normal user attempts to call this function, the transaction will automatically revert. By marking the `withdraw Ether` as payable, we can optimize gas costs for legitimate callers since the compiler will skip the checks for payme nt.

Bug ID	Bug Type	
SSB_318917_63	LONG REQUIRE/REVERT STRINGS	
Severity	**Action Taken**	**Detection Method**
● Gas	*Pending Fix*	Automated

Line No.	File Location
L22 - L22	contract.sol

Affected Code

contract.sol L22 - L22

```
        uint256 c = a * b;
        require(c / a == b, 'SafeMath: multiplication overflow');
        return c;
    }
```

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 wit h additional overhead for computing memory offset, and other parameters.

Remediation

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

Bug ID	Bug Type	
SSB_318917_64	**LONG REQUIRE/REVERT STRINGS**	

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

Line No.	File Location
L99 - L99	contract.sol

Affected Code

contract.sol L99 - L99

```
        uint256 currentAllowance = allowances[from][msg.sender];
        require(currentAllowance >= value, "Transfer amount exceeds allowance");
         allowances[from][msg.sender] = currentAllowance.sub(value);
         return true;
```

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 wit h additional overhead for computing memory offset, and other parameters.

Remediation

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

Bug ID
SSB_318917_65

Bug Type
LONG REQUIRE/REVERT STRINGS

Severity
● Gas

Action Taken
Pending Fix

Detection Method
Automated

Line No.
L151 - L151

File Location
contract.sol

Affected Code

contract.sol L151 - L151

```
        require(to != address(0), "Invalid address");
        require(value > 0, "Transfer value must be greater than zero");
        require(balances[from] >= value, "Insufficient balance");
```

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 wit
h additional overhead for computing memory offset, and other parameters.

Remediation

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

Bug ID

SSB_318917_66

Bug Type

LONG REQUIRE/REVERT STRINGS

Severity

● Gas

Action Taken

Pending Fix

Detection Method

Automated

Line No.

L195 - L195

File Location

contract.sol

Affected Code

contract.sol L195 - L195

```
        require(account != address(0), "Invalid address");
        require(lockDuration > 0, "Lock duration must be greater than zero");

        // Calculate the unlock timestamp based on the current block timestamp and the lock duration
```

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 wit h additional overhead for computing memory offset, and other parameters.

Remediation

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

Bug ID	Bug Type	
SSB_318917_67	**LONG REQUIRE/REVERT STRINGS**	

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

Line No.	File Location
L206 - L206	contract.sol

Affected Code

contract.sol L206 - L206

```
    function buyBYF(uint256 ethAmountInWei) external payable {
        require(ethAmountInWei > 0, "ETH amount must be greater than zero");

        // Implement mutex lock at the beginning of the function
```

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 wit h additional overhead for computing memory offset, and other parameters.

Remediation

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

Bug ID	Bug Type	
SSB_318917_68	LONG REQUIRE/REVERT STRINGS	
Severity	**Action Taken**	**Detection Method**
● Gas	⚠ *Pending Fix*	Automated

Line No.	File Location
L234 - L234	contract.sol

Affected Code

contract.sol L234 - L234

```
    function sellBYF(uint256 byfAmount) external {
        require(byfAmount > 0, "BYF amount must be greater than zero");
        require(balances[msg.sender] >= byfAmount, "Insufficient BYF balance");
```

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 wit h additional overhead for computing memory offset, and other parameters.

Remediation

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

Bug ID	Bug Type	
SSB_318917_15	**OPTIMIZING ADDRESS ID MAPPING**	
Severity	Action Taken	Detection Method
● Gas	*Pending Fix*	Automated

Line No.	File Location
L46 - L46	contract.sol

Affected Code

contract.sol L46 - L46

```
    mapping(address => uint256) private balances;
    mapping(address => mapping(address => uint256)) private allowances;
    mapping(address => uint256) private unlockTime;
```

Description

Combining multiple address/ID mappings into a single mapping using a struct enhances storage efficiency, simplifies code, and reduces gas costs, resulting in a more streamlined and cost-effective smart contract design.
It saves storage slot for the mapping and depending on the circumstances and sizes of types, it can avoid a Gsset (2 0000 gas) per mapping combined. Reads and subsequent writes can also be cheaper when a function requires both values and they fit in the same storage slot.

Remediation

It is suggested to modify the code so that multiple mappings using the address->id parameter are combined into a st ruct.

Bug ID	Bug Type	
SSB_318917_16	**OPTIMIZING ADDRESS ID MAPPING**	
Severity	Action Taken	Detection Method
● Gas	*Pending Fix*	Automated

Line No.	File Location
L47 - L47	contract.sol

Affected Code

contract.sol L47 - L47

```
    mapping(address => uint256) private balances;
    mapping(address => mapping(address => uint256)) private allowances;
    mapping(address => uint256) private unlockTime;
    mapping(address => bool) private mutex; // Mutex lock
```

Description

Combining multiple address/ID mappings into a single mapping using a struct enhances storage efficiency, simplifies code, and reduces gas costs, resulting in a more streamlined and cost-effective smart contract design.
It saves storage slot for the mapping and depending on the circumstances and sizes of types, it can avoid a Gsset (2 0000 gas) per mapping combined. Reads and subsequent writes can also be cheaper when a function requires both values and they fit in the same storage slot.

Remediation

It is suggested to modify the code so that multiple mappings using the address->id parameter are combined into a st ruct.

Bug ID	Bug Type	
SSB_318917_17	OPTIMIZING ADDRESS ID MAPPING	
Severity	**Action Taken**	**Detection Method**
● Gas	*Pending Fix*	Automated

Line No.	File Location
L48 - L48	contract.sol

Affected Code

contract.sol L48 - L48

```
    mapping(address => mapping(address => uint256)) private allowances;
    mapping(address => uint256) private unlockTime;
    mapping(address => bool) private mutex; // Mutex lock
```

Description

Combining multiple address/ID mappings into a single mapping using a struct enhances storage efficiency, simplifies code, and reduces gas costs, resulting in a more streamlined and cost-effective smart contract design.
It saves storage slot for the mapping and depending on the circumstances and sizes of types, it can avoid a Gsset (2 0000 gas) per mapping combined. Reads and subsequent writes can also be cheaper when a function requires both values and they fit in the same storage slot.

Remediation

It is suggested to modify the code so that multiple mappings using the address→id parameter are combined into a st ruct.

Bug ID	Bug Type
SSB_318917_18	OPTIMIZING ADDRESS ID MAPPING

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

Line No.	File Location
L49 - L49	contract.sol

Affected Code

contract.sol L49 - L49

```
    mapping(address => uint256) private unlockTime;
    mapping(address => bool) private mutex; // Mutex lock

    address payable public owner;
```

Description

Combining multiple address/ID mappings into a single mapping using a struct enhances storage efficiency, simplifies code, and reduces gas costs, resulting in a more streamlined and cost-effective smart contract design.
It saves storage slot for the mapping and depending on the circumstances and sizes of types, it can avoid a Gsset (2 0000 gas) per mapping combined. Reads and subsequent writes can also be cheaper when a function requires both values and they fit in the same storage slot.

Remediation

It is suggested to modify the code so that multiple mappings using the address->id parameter are combined into a st ruct.

Bug ID	Bug Type	
SSB_318917_72	**PUBLIC CONSTANTS CAN BE PRIVATE**	
Severity	Action Taken	Detection Method
● Gas	⚠ *Pending Fix*	Automated

Line No.	File Location
L36 - L36	contract.sol

Affected Code

contract.sol L36 - L36

```
    string public constant name = "BYFCOIN";
    string public constant symbol = "BYF";
    uint8 public constant decimals = 18;
```

Description

Public constant variables cost more gas because the EVM automatically creates getter functions for them and adds entries to the method ID table. The values can be read from the source code instead.
The following variable is affected: name

Remediation

If reading the values for the constants are not necessary, consider changing the `public` visibility to `private`.

Bug ID	Bug Type	
SSB_318917_73	**PUBLIC CONSTANTS CAN BE PRIVATE**	
Severity	Action Taken	Detection Method
● Gas	⚠ *Pending Fix*	Automated

Line No.	File Location
L37 - L37	contract.sol

Affected Code

contract.sol L37 - L37

```
string public constant name = "BYFCOIN";
string public constant symbol = "BYF";
uint8 public constant decimals = 18;
uint256 public totalSupply;
```

Description

Public constant variables cost more gas because the EVM automatically creates getter functions for them and adds entries to the method ID table. The values can be read from the source code instead.
The following variable is affected: symbol

Remediation

If reading the values for the constants are not necessary, consider changing the `public` visibility to `private`.

Bug ID	Bug Type	
SSB_318917_74	**PUBLIC CONSTANTS CAN BE PRIVATE**	
Severity	**Action Taken**	**Detection Method**
● Gas	⚠ *Pending Fix*	Automated

Line No.	File Location
L38 - L38	contract.sol

Affected Code

contract.sol L38 - L38

```
string public constant symbol = "BYF";
uint8 public constant decimals = 18;
uint256 public totalSupply;
uint256 public maxSupply;
```

Description

Public constant variables cost more gas because the EVM automatically creates getter functions for them and adds entries to the method ID table. The values can be read from the source code instead.
The following variable is affected: decimals

Remediation

If reading the values for the constants are not necessary, consider changing the `public` visibility to `private`.

Bug ID	Bug Type	
SSB_318917_1	**USE OF SAFEMATH LIBRARY**	
Severity	Action Taken	Detection Method
● Gas	⚠ *Pending Fix*	Automated

Line No.	File Location
L34 - L34	contract.sol

Affected Code

contract.sol L34 - L34

```
contract BYFCOIN {
    using SafeMath for uint256;

    string public constant name = "BYFCOIN";
```

Description

`SafeMath` library is found to be used in the contract. This increases gas consumption than traditional methods and validations if done manually.
Also, Solidity `0.8.0` includes checked arithmetic operations by default, and this renders `SafeMath` unnecessary.

Remediation

We do not recommend using `SafeMath` library for all arithmetic operations. It is good practice to use explicit checks where it is really needed and to avoid extra checks where overflow/underflow is impossible.
The compiler should be upgraded to Solidity version `0.8.0+` which automatically checks for overflows and underflo ws.

Bug ID	Bug Type	
SSB_318917_35	SMALLER DATA TYPES COST MORE	
Severity	**Action Taken**	**Detection Method**
● Gas	*Pending Fix*	Automated

Line No.	File Location
L70 - L70	contract.sol

Affected Code

contract.sol L70 - L70

```
        owner = payable(msg.sender); // Set the owner to the address that deploys the contract
        totalSupply = 1000000000 * 10 ** uint256(decimals);
        maxSupply = totalSupply;
        maxWalletBalance = 20000 * 10 ** uint256(decimals);
```

Description

Usage of smaller integer types such as `uint8`, `uint16`, `int8`, or `int16` in arithmetic operations incur additional gas costs compared to the default `uint` and `int` types, which are typically `uint256` and `int256` respectively.

Remediation

Replace occurrences of smaller integer types (`uint8`, `uint16`, `int8`, `int16`) with the default integer types (`uint` or `int`). This can be achieved by simply using `uint` or `int`, which are automatically mapped to `uint256` and `int256` respectively in Solidity version `0.8.0` and above.

Bug ID

SSB_318917_36

Bug Type

SMALLER DATA TYPES COST MORE

Severity

● Gas

Action Taken

Pending Fix

Detection Method

Automated

Line No.

L72 - L72

File Location

contract.sol

Affected Code

contract.sol L72 - L72

```
        maxSupply = totalSupply;
        maxWalletBalance = 20000 * 10 ** uint256(decimals);
        lockTimeBlocks = 105120000; // Equivalent to approximately 2 years with 15 seconds per block
        rate = 100000; // Initial rate: 100000 BYF per 1 ETH
```

Description

Usage of smaller integer types such as `uint8`, `uint16`, `int8`, or `int16` in arithmetic operations incur additional gas costs compared to the default `uint` and `int` types, which are typically `uint256` and `int256` respectively.

Remediation

Replace occurrences of smaller integer types (`uint8`, `uint16`, `int8`, `int16`) with the default integer types (`uint` or `int`). This can be achieved by simply using `uint` or `int`, which are automatically mapped to `uint256` and `int256` respectively in Solidity version `0.8.0` and above.

Bug ID

SSB_318917_37

Bug Type

SMALLER DATA TYPES COST MORE

Severity

● Gas

Action Taken

Pending Fix

Detection Method

Automated

Line No.

L79 - L79

File Location

contract.sol

Affected Code

contract.sol L79 - L79

```
        // Lock a portion of the owner's wallet balance for 2 years
        uint256 lockedBalance = 100000000 * 10 ** uint256(decimals);
        _lockTokens(msg.sender, lockedBalance, lockTimeBlocks);
```

Description

Usage of smaller integer types such as `uint8`, `uint16`, `int8`, or `int16` in arithmetic operations incur additional gas costs compared to the default `uint` and `int` types, which are typically `uint256` and `int256` respectively.

Remediation

Replace occurrences of smaller integer types (`uint8`, `uint16`, `int8`, `int16`) with the default integer types (`uint` or `int`). This can be achieved by simply using `uint` or `int`, which are automatically mapped to `uint256` and `int256` respectively in Solidity version `0.8.0` and above.

Bug ID	Bug Type	
SSB_318917_38	SMALLER DATA TYPES COST MORE	
Severity	**Action Taken**	**Detection Method**
● Gas	*Pending Fix*	Automated

Line No.	File Location
L84 - L84	contract.sol

Affected Code

contract.sol L84 - L84

```
        tradingAddress = address(this);
        balances[tradingAddress] = 100000000 * 10 ** uint256(decimals);
    }
```

Description

Usage of smaller integer types such as `uint8`, `uint16`, `int8`, or `int16` in arithmetic operations incur additional gas costs compared to the default `uint` and `int` types, which are typically `uint256` and `int256` respectively.

Remediation

Replace occurrences of smaller integer types (`uint8`, `uint16`, `int8`, `int16`) with the default integer types (`uint` or `int`). This can be achieved by simply using `uint` or `int`, which are automatically mapped to `uint256` and `int256` respectively in Solidity version `0.8.0` and above.

Bug ID	Bug Type	
SSB_318917_39	**STORAGE VARIABLE CACHING IN MEMORY**	
Severity	Action Taken	Detection Method
● Gas	*Pending Fix*	Automated

Line No.	File Location
L68 - L85	contract.sol

Affected Code

contract.sol L68 - L85

```

    constructor() {
        owner = payable(msg.sender); // Set the owner to the address that deploys the contract
        totalSupply = 1000000000 * 10 ** uint256(decimals);
       maxSupply = totalSupply;
        maxWalletBalance = 20000 * 10 ** uint256(decimals);
        lockTimeBlocks = 105120000; // Equivalent to approximately 2 years with 15 seconds per block
        rate = 100000; // Initial rate: 100000 BYF per 1 ETH

        balances[msg.sender] = totalSupply;

        // Lock a portion of the owner's wallet balance for 2 years
        uint256 lockedBalance = 100000000 * 10 ** uint256(decimals);
        _lockTokens(msg.sender, lockedBalance, lockTimeBlocks);

        // Allocate 100,000,000 BYF for trading
        tradingAddress = address(this);
        balances[tradingAddress] = 100000000 * 10 ** uint256(decimals);
    }

    function balanceOf(address account) external view returns (uint256) {
        return balances[account];
```

Description

The contract `BYFCOIN` is using the state variable `decimals` multiple times in the function ` `.
`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 (costin g 1 `SLOAD`) and then read from this cache to avoid multiple `SLOADs`.

Bug ID	Bug Type
SSB_318917_39	STORAGE VARIABLE CACHING IN MEMORY

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

Line No.	File Location
L68 - L85	contract.sol

Affected Code

contract.sol L68 - L85

```
    constructor() {
        owner = payable(msg.sender); // Set the owner to the address that deploys the contract
        totalSupply = 1000000000 * 10 ** uint256(decimals);
        maxSupply = totalSupply;
        maxWalletBalance = 20000 * 10 ** uint256(decimals);
        lockTimeBlocks = 105120000; // Equivalent to approximately 2 years with 15 seconds per block
        rate = 100000; // Initial rate: 100000 BYF per 1 ETH

        balances[msg.sender] = totalSupply;

        // Lock a portion of the owner's wallet balance for 2 years
        uint256 lockedBalance = 100000000 * 10 ** uint256(decimals);
        _lockTokens(msg.sender, lockedBalance, lockTimeBlocks);

        // Allocate 100,000,000 BYF for trading
        tradingAddress = address(this);
        balances[tradingAddress] = 100000000 * 10 ** uint256(decimals);
    }

    function balanceOf(address account) external view returns (uint256) {
        return balances[account];
```

Description

The contract `BYFCOIN` is using the state variable `balances` multiple times in the function ` `.
`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 (costin g 1 `SLOAD`) and then read from this cache to avoid multiple `SLOADs`.

Bug ID	Bug Type	
SSB_318917_46	**STORAGE VARIABLE CACHING IN MEMORY**	
Severity	Action Taken	Detection Method
● Gas	*Pending Fix*	Automated

Line No.	File Location
L96 - L102	contract.sol

Affected Code

contract.sol L96 - L102

```
    function transferFrom(address from, address to, uint256 value) external returns (bool) {
        _transfer(from, to, value);
        uint256 currentAllowance = allowances[from][msg.sender];
        require(currentAllowance >= value, "Transfer amount exceeds allowance");
        allowances[from][msg.sender] = currentAllowance.sub(value);
        return true;
    }

    function approve(address spender, uint256 value) external returns (bool) {
```

Description

The contract `BYFCOIN` is using the state variable `allowances` multiple times in the function `transferFrom`. `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 (costin g 1 `SLOAD`) and then read from this cache to avoid multiple `SLOADs`.

Bug ID	Bug Type	
SSB_318917_47	**STORAGE VARIABLE CACHING IN MEMORY**	
Severity	Action Taken	Detection Method
● Gas	*Pending Fix*	Automated

Line No.	File Location
L110 - L114	contract.sol

Affected Code

contract.sol L110 - L114

```
    function increaseAllowance(address spender, uint256 addedValue) external returns (bool) {
        allowances[msg.sender][spender] = allowances[msg.sender][spender].add(addedValue);
        emit Approval(msg.sender, spender, allowances[msg.sender][spender]);
        return true;
    }

    function decreaseAllowance(address spender, uint256 subtractedValue) external returns (bool) {
```

Description

The contract `BYFCOIN` is using the state variable `allowances` multiple times in the function `increaseAllowance`.
`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

SSB_318917_48

Bug Type

STORAGE VARIABLE CACHING IN MEMORY

Severity

● Gas

Action Taken

Pending Fix

Detection Method

Automated

Line No.

L116 - L122

File Location

contract.sol

Affected Code

contract.sol L116 - L122

```
    function decreaseAllowance(address spender, uint256 subtractedValue) external returns (bool) {
        uint256 currentAllowance = allowances[msg.sender][spender];
        require(currentAllowance >= subtractedValue, "Decreased allowance below zero");
        allowances[msg.sender][spender] = currentAllowance.sub(subtractedValue);
        emit Approval(msg.sender, spender, allowances[msg.sender][spender]);
        return true;
    }

    function withdrawTokens(uint256 amount) external onlyOwner {
```

Description

The contract `BYFCOIN` is using the state variable `allowances` multiple times in the function `decreaseAllowance`.
`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

SSB_318917_49

Bug Type

STORAGE VARIABLE CACHING IN MEMORY

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

Line No.	File Location
L128 - L133	contract.sol

Affected Code

contract.sol L128 - L133

```

    function withdrawEther(uint256 amount) external onlyOwner {
        require(amount <= address(this).balance, "Insufficient contract balance");

        owner.transfer(amount); // Transfer the specified amount to the owner
        emit Withdraw(owner, amount); // Emit withdrawal event
    }

    function isUnlocked(address account) external view returns (bool) {
```

Description

The contract `BYFCOIN` is using the state variable `owner` multiple times in the function `withdrawEther`. `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 (costin g 1 `SLOAD`) and then read from this cache to avoid multiple `SLOADs`.

Bug ID	Bug Type	
SSB_318917_50	**STORAGE VARIABLE CACHING IN MEMORY**	
Severity	Action Taken	Detection Method
● Gas	*Pending Fix*	Automated

Line No.	File Location
L145 - L182	contract.sol

Affected Code

contract.sol L145 - L182

```
    // Internal transfer function
    function _transfer(address from, address to, uint256 value) private {
        // Implement mutex lock at the beginning of the function
        require(!mutex[from], "Transfer in progress");
        mutex[from] = true;

        require(to != address(0), "Invalid address");
        require(value > 0, "Transfer value must be greater than zero");
        require(balances[from] >= value, "Insufficient balance");

        // Calculate the tax amount based on the tax rate
        uint256 taxAmount = (value.mul(taxRate)).div(100);

        // Deduct tax from transfer amount
        uint256 transferAmount = value.sub(taxAmount);

        if (from != owner && to != owner && balances[to].add(transferAmount) > maxWalletBalance) {
            uint256 excessTokens = balances[to].add(transferAmount).sub(maxWalletBalance);
            _lockTokens(to, excessTokens, lockTimeBlocks);
            transferAmount = transferAmount.sub(excessTokens);
        }

        balances[from] = balances[from].sub(value);
        balances[to] = balances[to].add(transferAmount);

        if (taxAmount > 0) {
            // Convert tax amount to ETH
            uint256 ethTaxAmount = _calculateEthAmount(taxAmount);
```

contract.sol L145 - L182

```
            // Convert tax amount to ETH
            uint256 ethTaxAmount = _calculateEthAmount(taxAmount);
            // Transfer ETH tax to owner's wallet
            owner.transfer(ethTaxAmount);
            emit Transfer(from, owner, taxAmount);
            emit TaxDeducted(from, owner, ethTaxAmount); // Emit tax deduction event
        }

        emit Transfer(from, to, transferAmount);

        // Clear mutex lock at the end of the function
        mutex[from] = false;
    }

    // Function to calculate ETH amount equivalent to given BYF amount
```

Description

The contract `BYFCOIN` is using the state variable `maxWalletBalance` multiple times in the function `_transfer`. `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 (costin g 1 `SLOAD`) and then read from this cache to avoid multiple `SLOADs`.

Bug ID	Bug Type	
SSB_318917_50	STORAGE VARIABLE CACHING IN MEMORY	
Severity	**Action Taken**	**Detection Method**
● Gas	*Pending Fix*	Automated

Line No.	File Location
L145 - L182	contract.sol

Affected Code

contract.sol L145 - L182

```
    // Internal transfer function
    function _transfer(address from, address to, uint256 value) private {
        // Implement mutex lock at the beginning of the function
        require(!mutex[from], "Transfer in progress");
        mutex[from] = true;

        require(to != address(0), "Invalid address");
        require(value > 0, "Transfer value must be greater than zero");
        require(balances[from] >= value, "Insufficient balance");

        // Calculate the tax amount based on the tax rate
        uint256 taxAmount = (value.mul(taxRate)).div(100);

        // Deduct tax from transfer amount
        uint256 transferAmount = value.sub(taxAmount);

        if (from != owner && to != owner && balances[to].add(transferAmount) > maxWalletBalance) {
            uint256 excessTokens = balances[to].add(transferAmount).sub(maxWalletBalance);
            _lockTokens(to, excessTokens, lockTimeBlocks);
            transferAmount = transferAmount.sub(excessTokens);
        }

        balances[from] = balances[from].sub(value);
        balances[to] = balances[to].add(transferAmount);

        if (taxAmount > 0) {
            // Convert tax amount to ETH
            uint256 ethTaxAmount = _calculateEthAmount(taxAmount);
```

contract.sol L145 - L182

```
                // Convert tax amount to ETH
                uint256 ethTaxAmount = _calculateEthAmount(taxAmount);
                // Transfer ETH tax to owner's wallet
                owner.transfer(ethTaxAmount);
                emit Transfer(from, owner, taxAmount);
                emit TaxDeducted(from, owner, ethTaxAmount); // Emit tax deduction event
            }

            emit Transfer(from, to, transferAmount);

            // Clear mutex lock at the end of the function
            mutex[from] = false;
        }

        // Function to calculate ETH amount equivalent to given BYF amount
```

Description

The contract `BYFCOIN` is using the state variable `balances` multiple times in the function `_transfer`. `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 (costin g 1 `SLOAD`) and then read from this cache to avoid multiple `SLOADs`.

Bug ID

SSB_318917_50

Bug Type

STORAGE VARIABLE CACHING IN MEMORY

Severity

● Gas

Action Taken

Pending Fix

Detection Method

Automated

Line No.

L145 - L182

File Location

contract.sol

Affected Code

contract.sol L145 - L182

```
    // Internal transfer function
    function _transfer(address from, address to, uint256 value) private {
        // Implement mutex lock at the beginning of the function
        require(!mutex[from], "Transfer in progress");
        mutex[from] = true;

        require(to != address(0), "Invalid address");
        require(value > 0, "Transfer value must be greater than zero");
        require(balances[from] >= value, "Insufficient balance");

        // Calculate the tax amount based on the tax rate
        uint256 taxAmount = (value.mul(taxRate)).div(100);

        // Deduct tax from transfer amount
        uint256 transferAmount = value.sub(taxAmount);

        if (from != owner && to != owner && balances[to].add(transferAmount) > maxWalletBalance) {
            uint256 excessTokens = balances[to].add(transferAmount).sub(maxWalletBalance);
            _lockTokens(to, excessTokens, lockTimeBlocks);
            transferAmount = transferAmount.sub(excessTokens);
        }

        balances[from] = balances[from].sub(value);
        balances[to] = balances[to].add(transferAmount);

        if (taxAmount > 0) {
            // Convert tax amount to ETH
            uint256 ethTaxAmount = _calculateEthAmount(taxAmount);
```

contract.sol L145 - L182

```
            // Convert tax amount to ETH
            uint256 ethTaxAmount = _calculateEthAmount(taxAmount);
            // Transfer ETH tax to owner's wallet
            owner.transfer(ethTaxAmount);
            emit Transfer(from, owner, taxAmount);
            emit TaxDeducted(from, owner, ethTaxAmount); // Emit tax deduction event
        }

        emit Transfer(from, to, transferAmount);

        // Clear mutex lock at the end of the function
        mutex[from] = false;
    }

    // Function to calculate ETH amount equivalent to given BYF amount
```

Description

The contract `BYFCOIN` is using the state variable `mutex` multiple times in the function `_transfer`. `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

SSB_318917_50

Bug Type

STORAGE VARIABLE CACHING IN MEMORY

Severity

● Gas

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L145 - L182

File Location

contract.sol

Affected Code

contract.sol

L145 - L182

```
    // Internal transfer function
    function _transfer(address from, address to, uint256 value) private {
        // Implement mutex lock at the beginning of the function
        require(!mutex[from], "Transfer in progress");
        mutex[from] = true;

        require(to != address(0), "Invalid address");
        require(value > 0, "Transfer value must be greater than zero");
        require(balances[from] >= value, "Insufficient balance");

        // Calculate the tax amount based on the tax rate
        uint256 taxAmount = (value.mul(taxRate)).div(100);

        // Deduct tax from transfer amount
        uint256 transferAmount = value.sub(taxAmount);

        if (from != owner && to != owner && balances[to].add(transferAmount) > maxWalletBalance) {
            uint256 excessTokens = balances[to].add(transferAmount).sub(maxWalletBalance);
            _lockTokens(to, excessTokens, lockTimeBlocks);
            transferAmount = transferAmount.sub(excessTokens);
        }

        balances[from] = balances[from].sub(value);
        balances[to] = balances[to].add(transferAmount);

        if (taxAmount > 0) {
            // Convert tax amount to ETH
            uint256 ethTaxAmount = _calculateEthAmount(taxAmount);
```

contract.sol L145 - L182

```
                // Convert tax amount to ETH
                uint256 ethTaxAmount = _calculateEthAmount(taxAmount);
                // Transfer ETH tax to owner's wallet
                owner.transfer(ethTaxAmount);
                emit Transfer(from, owner, taxAmount);
                emit TaxDeducted(from, owner, ethTaxAmount); // Emit tax deduction event
            }

            emit Transfer(from, to, transferAmount);

            // Clear mutex lock at the end of the function
            mutex[from] = false;
        }

        // Function to calculate ETH amount equivalent to given BYF amount
```

Description

The contract `BYFCOIN` is using the state variable `owner` multiple times in the function `_transfer`. `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 (costin g 1 `SLOAD`) and then read from this cache to avoid multiple `SLOADs`.

Bug ID

SSB_318917_78

Bug Type

STORAGE VARIABLE CACHING IN MEMORY

Severity

● Gas

Action Taken

Pending Fix

Detection Method

Automated

Line No.

L185 - L190

File Location

contract.sol

Affected Code

contract.sol L185 - L190

```
    // Function to calculate ETH amount equivalent to given BYF amount
    function _calculateEthAmount(uint256 byfAmount) private view returns (uint256) {
        require(rate > 0, "Rate must be greater than zero");
        // Calculate ETH amount based on current rate
        uint256 ethAmount = byfAmount.div(rate);
        return ethAmount;
    }

    // Lock tokens for the specified duration using a timestamp
```

Description

The contract `BYFCOIN` is using the state variable `rate` multiple times in the function `_calculateEthAmount`. `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 (costin g 1 `SLOAD`) and then read from this cache to avoid multiple `SLOADs`.

Bug ID

SSB_318917_79

Bug Type

STORAGE VARIABLE CACHING IN MEMORY

Severity

● Gas

Action Taken

⚠ *Pending Fix*

Detection Method

Automated

Line No.

L193 - L203

File Location

contract.sol

Affected Code

contract.sol L193 - L203

```
    // Lock tokens for the specified duration using a timestamp
    function _lockTokens(address account, uint256 amount, uint256 lockDuration) private {
        require(account != address(0), "Invalid address");
        require(lockDuration > 0, "Lock duration must be greater than zero");

        // Calculate the unlock timestamp based on the current block timestamp and the lock duration
        uint256 unlockTimestamp = block.timestamp + lockDuration;

        unlockTime[account] = unlockTimestamp;
        balances[account] = balances[account].sub(amount);
        emit Transfer(account, address(0), amount); // Event emitted after state change
    }

    function buyBYF(uint256 ethAmountInWei) external payable {
        require(ethAmountInWei > 0, "ETH amount must be greater than zero");
```

Description

The contract `BYFCOIN` is using the state variable `balances` multiple times in the function `_lockTokens`. `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 (costin g 1 `SLOAD`) and then read from this cache to avoid multiple `SLOADs`.

Bug ID
SSB_318917_80

Bug Type
STORAGE VARIABLE CACHING IN MEMORY

Severity
● Gas

Action Taken
Pending Fix

Detection Method
Automated

Line No.
L205 - L230

File Location
contract.sol

Affected Code

contract.sol L205 - L230

```
    function buyBYF(uint256 ethAmountInWei) external payable {
        require(ethAmountInWei > 0, "ETH amount must be greater than zero");

        // Implement mutex lock at the beginning of the function
        require(!mutex[msg.sender], "Buy in progress");
        mutex[msg.sender] = true;

        // Calculate the amount of BYF tokens to be bought based on the provided ETH amount and the
current rate
        uint256 byfAmount = ethAmountInWei.mul(rate); // Convert from wei to BYF

        // Ensure that the contract has enough BYF tokens to fulfill the purchase
        require(balances[tradingAddress] >= byfAmount, "Insufficient BYF balance");

        // Transfer BYF tokens to the buyer
        balances[msg.sender] = balances[msg.sender].add(byfAmount);
        balances[tradingAddress] = balances[tradingAddress].sub(byfAmount);

        // Emit the Bought event
        emit Bought(msg.sender, byfAmount, ethAmountInWei);

        // Update the rate
        _updateRate(true);

        // Clear mutex lock at the end of the function
        mutex[msg.sender] = false;
    }

    // Function to sell BYF tokens for ETH
```

Description

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

Remediation

Bug ID

SSB_318917_80

Bug Type

STORAGE VARIABLE CACHING IN MEMORY

Severity

● Gas

Action Taken

Pending Fix

Detection Method

Automated

Line No.

L205 - L230

File Location

contract.sol

Affected Code

contract.sol L205 - L230

```
    function buyBYF(uint256 ethAmountInWei) external payable {
        require(ethAmountInWei > 0, "ETH amount must be greater than zero");

        // Implement mutex lock at the beginning of the function
        require(!mutex[msg.sender], "Buy in progress");
        mutex[msg.sender] = true;

        // Calculate the amount of BYF tokens to be bought based on the provided ETH amount and the
current rate
        uint256 byfAmount = ethAmountInWei.mul(rate); // Convert from wei to BYF

        // Ensure that the contract has enough BYF tokens to fulfill the purchase
        require(balances[tradingAddress] >= byfAmount, "Insufficient BYF balance");

        // Transfer BYF tokens to the buyer
        balances[msg.sender] = balances[msg.sender].add(byfAmount);
        balances[tradingAddress] = balances[tradingAddress].sub(byfAmount);

        // Emit the Bought event
        emit Bought(msg.sender, byfAmount, ethAmountInWei);

        // Update the rate
        _updateRate(true);

        // Clear mutex lock at the end of the function
        mutex[msg.sender] = false;
    }

    // Function to sell BYF tokens for ETH
```

Description

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

Remediation

Bug ID	Bug Type	
SSB_318917_80	STORAGE VARIABLE CACHING IN MEMORY	
Severity	**Action Taken**	**Detection Method**
● Gas	⚠ *Pending Fix*	Automated

Line No.	File Location
L205 - L230	contract.sol

Affected Code

contract.sol L205 - L230

```
    function buyBYF(uint256 ethAmountInWei) external payable {
        require(ethAmountInWei > 0, "ETH amount must be greater than zero");

        // Implement mutex lock at the beginning of the function
        require(!mutex[msg.sender], "Buy in progress");
        mutex[msg.sender] = true;

        // Calculate the amount of BYF tokens to be bought based on the provided ETH amount and the
current rate
        uint256 byfAmount = ethAmountInWei.mul(rate); // Convert from wei to BYF

        // Ensure that the contract has enough BYF tokens to fulfill the purchase
        require(balances[tradingAddress] >= byfAmount, "Insufficient BYF balance");

        // Transfer BYF tokens to the buyer
        balances[msg.sender] = balances[msg.sender].add(byfAmount);
        balances[tradingAddress] = balances[tradingAddress].sub(byfAmount);

        // Emit the Bought event
        emit Bought(msg.sender, byfAmount, ethAmountInWei);

        // Update the rate
        _updateRate(true);

        // Clear mutex lock at the end of the function
        mutex[msg.sender] = false;
    }

    // Function to sell BYF tokens for ETH
```

Description

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

Remediation

Bug ID	Bug Type
SSB_318917_81	STORAGE VARIABLE CACHING IN MEMORY

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

Line No.	File Location
L233 - L258	contract.sol

Affected Code

contract.sol L233 - L258

```
    function sellBYF(uint256 byfAmount) external {
        require(byfAmount > 0, "BYF amount must be greater than zero");
        require(balances[msg.sender] >= byfAmount, "Insufficient BYF balance");

        // Implement mutex lock at the beginning of the function
        require(!mutex[msg.sender], "Sell in progress");
        mutex[msg.sender] = true;

        // Calculate the amount of ETH to be received based on the current rate
        uint256 ethAmount = byfAmount.div(rate);

        // Transfer BYF tokens from the seller
        balances[msg.sender] = balances[msg.sender].sub(byfAmount);

        // Transfer ETH to the seller
        payable(msg.sender).transfer(ethAmount);

        // Emit the Sold event
        emit Sold(msg.sender, byfAmount, ethAmount);

        // Update the rate
        _updateRate(false);

        // Clear mutex lock at the end of the function
        mutex[msg.sender] = false;
    }

    // Internal function to update the rate
```

Description

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

Remediation

Bug ID

SSB_318917_81

Bug Type

STORAGE VARIABLE CACHING IN MEMORY

Severity

● Gas

Action Taken

Pending Fix

Detection Method

Automated

Line No.

L233 - L258

File Location

contract.sol

Affected Code

contract.sol L233 - L258

```
    function sellBYF(uint256 byfAmount) external {
        require(byfAmount > 0, "BYF amount must be greater than zero");
        require(balances[msg.sender] >= byfAmount, "Insufficient BYF balance");

        // Implement mutex lock at the beginning of the function
        require(!mutex[msg.sender], "Sell in progress");
        mutex[msg.sender] = true;

        // Calculate the amount of ETH to be received based on the current rate
        uint256 ethAmount = byfAmount.div(rate);

        // Transfer BYF tokens from the seller
        balances[msg.sender] = balances[msg.sender].sub(byfAmount);

        // Transfer ETH to the seller
        payable(msg.sender).transfer(ethAmount);

        // Emit the Sold event
        emit Sold(msg.sender, byfAmount, ethAmount);

        // Update the rate
        _updateRate(false);

        // Clear mutex lock at the end of the function
        mutex[msg.sender] = false;
    }

    // Internal function to update the rate
```

Description

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

Remediation

Bug ID

SSB_318917_82

Bug Type

STORAGE VARIABLE CACHING IN MEMORY

Severity

● Gas

Action Taken

Pending Fix

Detection Method

Automated

Line No.

L261 - L272

File Location

contract.sol

Affected Code

contract.sol L261 - L272

```
    // Internal function to update the rate
    function _updateRate(bool isBuy) private {
        if (isBuy) {
            // Decrease rate by 0.4% after each buy
            rate = rate.mul(996).div(1000);
        } else {
            // Increase rate by 0.1% after each sell
            rate = rate.mul(1001).div(1000);
        }

        // Emit the RateUpdated event with the new rate
        emit RateUpdated(rate);
    }
}
```

Description

The contract `BYFCOIN` is using the state variable `rate` multiple times in the function `_updateRate`. `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 (costin
g 1 `SLOAD`) and then read from this cache to avoid multiple `SLOADs`.

Bug ID
SSB_318917_33

Bug Type
USE SELFBALANCE() INSTEAD OF ADDRESS(THIS).BALANCE

Severity
● Gas

Action Taken
Pending Fix

Detection Method
Automated

Line No.
L129 - L129

File Location
contract.sol

Affected Code

contract.sol L129 - L129

```
    function withdrawEther(uint256 amount) external onlyOwner {
        require(amount <= address(this).balance, "Insufficient contract balance");

        owner.transfer(amount); // Transfer the specified amount to the owner
```

Description

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

Remediation

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

Bug ID

SSB_318917_70

Bug Type

VARIABLES DECLARED BUT NEVER USED

Severity

● Gas

Action Taken

Pending Fix

Detection Method

Automated

Line No.

L36 - L36

File Location

contract.sol

Affected Code

contract.sol L36 - L36

```
        string public constant name = "BYFCOIN";
        string public constant symbol = "BYF";
        uint8 public constant decimals = 18;
```

Description

The contract BYFCOIN has declared a variable name but it is not used anywhere in the code. This represents dead c ode or missing logic.
Unused variables increase the contract's size and complexity, potentially leading to higher gas costs and a larger att ack surface.

Remediation

To remediate this vulnerability, developers should perform a code review and remove any variables that are declared but never used.

Bug ID	Bug Type	
SSB_318917_71	**VARIABLES DECLARED BUT NEVER USED**	
Severity	Action Taken	Detection Method
● Gas	*Pending Fix*	Automated
Line No.	File Location	
L37 - L37	contract.sol	

Affected Code

contract.sol L37 - L37

```
        string public constant name = "BYFCOIN";
        string public constant symbol = "BYF";
        uint8 public constant decimals = 18;
        uint256 public totalSupply;
```

Description

The contract BYFCOIN has declared a variable symbol but it is not used anywhere in the code. This represents dead code or missing logic.
Unused variables increase the contract's size and complexity, potentially leading to higher gas costs and a larger att ack surface.

Remediation

To remediate this vulnerability, developers should perform a code review and remove any variables that are declared but never used.

5. **Scan** History

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

No	Date	Security Score	Scan Overview
1.	2024-05-09	**80.55**	● 0 ● 0 ● 0 ● 5 ● 28 ● 54

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 SolidityScan 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. SolidityScan 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.