Cyberscope

Audit Report
SafeBit

December 2023

Network	BSC
Address	0x9fDE46Cb18E295c7a33a1Bdde9faA8a3927c010C
Audited by	© cyberscope

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	PVC	Price Volatility Concern	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Review

Contract Name	SafeBit
Compiler Version	v0.8.17+commit.8df45f5f
Optimization	200 runs
Explorer	https://bscscan.com/address/0x9fde46cb18e295c7a33a1bdde9faa8a3927c010c
Address	0x9fde46cb18e295c7a33a1bdde9faa8a3927c010c
Network	BSC
Symbol	SFBT
Decimals	9
Total Supply	5,000,000,000

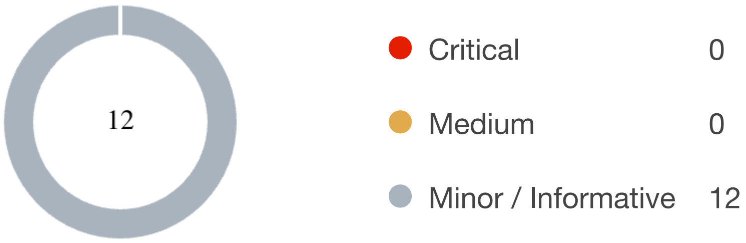
Audit Updates

Initial Audit	30 Dec 2023

Source Files

Filename	SHA256
SafeBit.sol	d753a4158a5ae57d25ea40e7bc75ee53550f1aa6d4a0456d8848863ddc774446

Findings Breakdown



Severity	Unresolved	Acknowledged	Resolved	Other
● Critical	0	0	0	0
● Medium	0	0	0	0
● Minor / Informative	12	0	0	0

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	SafeBit.sol#L547
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `swapTrigger` sets a threshold where the contract will trigger the swap functionality. If the variable is set to a big number, then the contract will swap a huge amount of tokens for ETH.

It is important to note that the price of the token representing it, can be highly volatile. This means that the value of a price volatility swap involving Ether could fluctuate significantly at the triggered point, potentially leading to significant price volatility for the parties involved.

```
function set_Number_Of_Transactions_Before_Liquify_Trigger(uint8
number_of_transactions) public onlyOwner {
    swapTrigger = number_of_transactions;
}
```

Recommendation

The contract could ensure that it will not sell more than a reasonable amount of tokens in a single transaction. A suggested implementation could check that the maximum amount should be less than a fixed percentage of the exchange reserves. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	SafeBit.sol#L529,590,600,823,824,830,835
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
Wallet_Dev = wallet;
noBlackList = true_or_false;
noFeeToTransfer = true_or_false;
uniswapV2Router = _newPCSRouter;
...
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	SafeBit.sol
Status	Unresolved

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert to underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases gas consumption unnecessarily.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	SafeBit.sol#L368,369,372,373,374,375,376,384
Status	Unresolved

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
address payable private Wallet_Burn =
payable(0x000000000000000000000000000000000000dEaD)
address payable private Wallet_zero =
payable(0x0000000000000000000000000000000000000000)
string private _name = "SafeBit"
string private _symbol = "SFBT"
uint8 private _decimals = 9
uint256 private _tTotal = 5000000000 * 10**9
uint256 private _tFeeTotal
uint256 private maxPossibleFee = 100
```

Recommendation

Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	SafeBit.sol#L192,193,206,223,362,363,367,368,369,388,389,390,404,406,519,528,541,547,556,574,589,599,621,626,772,805,821,828,834
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
function DOMAIN_SEPARATOR() external view returns (bytes32);
function PERMIT_TYPEHASH() external pure returns (bytes32);
function MINIMUM_LIQUIDITY() external pure returns (uint);
function WETH() external pure returns (address);
mapping (address => bool) public _isExcludedFromFee
mapping (address => bool) public _isBlacklisted
address payable private Wallet_Dev =
payable(0x256Ab2e0EfFe92e5229cDc006aB4d20E69Dc32FC)
address payable private Wallet_Burn =
payable(0x000000000000000000000000000000000000dEaD)
address payable private Wallet_zero =
payable(0x0000000000000000000000000000000000000000)
uint256 private _TotalFee = 3
uint256 public _buyFee = 1
uint256 public _sellFee = 2
uint256 public _maxWalletToken = _tTotal.mul(4).div(100)
uint256 public _maxTxAmount = _tTotal.mul(4).div(100)

...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.
Find more information on the Solidity documentation
https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L05 - Unused State Variable

Criticality	Minor / Informative
Location	SafeBit.sol#L369,376,405,407
Status	Unresolved

Description

An unused state variable is a state variable that is declared in the contract, but is never used in any of the contract's functions. This can happen if the state variable was originally intended to be used, but was later removed or never used.

Unused state variables can create clutter in the contract and make it more difficult to understand and maintain. They can also increase the size of the contract and the cost of deploying and interacting with it.

```
address payable private Wallet_zero =
payable(0x0000000000000000000000000000000000000000)
uint256 private _tFeeTotal
uint256 private _previousMaxWalletToken = _maxWalletToken
uint256 private _previousMaxTxAmount = _maxTxAmount
```

Recommendation

To avoid creating unused state variables, it's important to carefully consider the state variables that are needed for the contract's functionality, and to remove any that are no longer needed. This can help improve the clarity and efficiency of the contract.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	SafeBit.sol#L522,548,622,627
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
_sellFee = Sell_Fee
swapTrigger = number_of_transactions
_maxTxAmount = _tTotal*maxTxPercent_x100/10000
_maxWalletToken = _tTotal*maxWallPercent_x100/10000
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	SafeBit.sol#L64,70,76,80,84,88,95,99,106,110,116
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function isContract(address account) internal view returns (bool) {
        uint256 size;
        assembly { size := extcodesize(account) }
        return size > 0;
    }

...
        (bool success, ) = recipient.call{ value: amount }("");
        require(success, "Address: unable to send value, recipient may
have reverted");
    }

function functionCall(address target, bytes memory data) internal returns
(bytes memory) {
      return functionCall(target, data, "Address: low-level call failed");
    }

...
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	SafeBit.sol#L559,577
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
uint256 gasUsed
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	SafeBit.sol#L529,835
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
Wallet_Dev = wallet
uniswapV2Pair = newPair
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	SafeBit.sol#L66,121
Status	Unresolved

Description

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

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

```
assembly { size := extcodesize(account) }

assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
```

Recommendation

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

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	SafeBit.sol#L2
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.17;
```

Recommendation

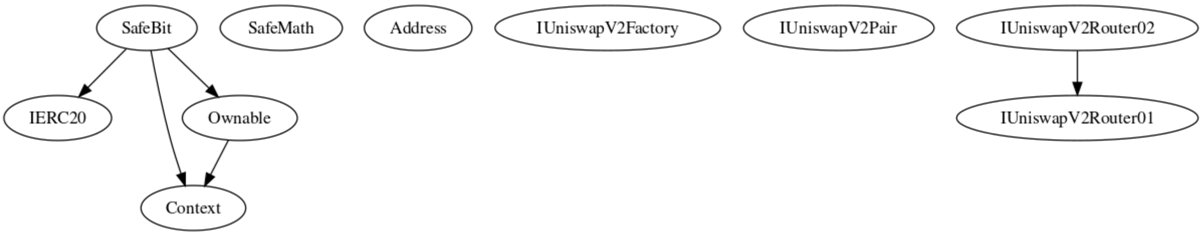
The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

Functions Analysis

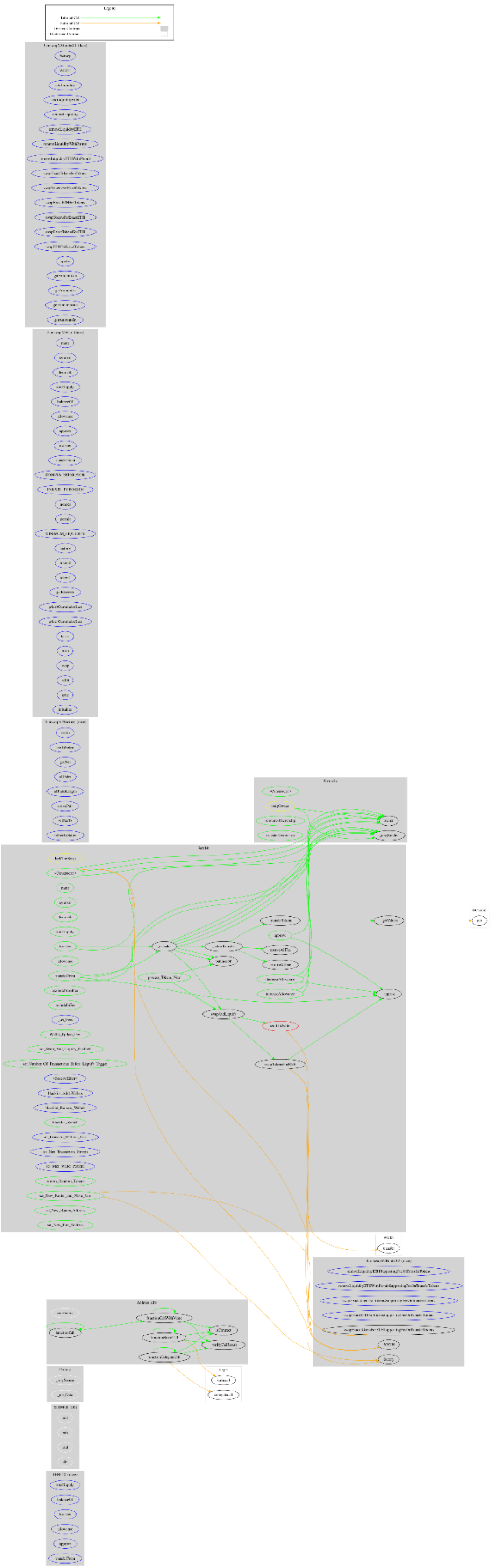
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
SafeBit	Implementation	Context, IERC20, Ownable		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	excludeFromFee	Public	✓	onlyOwner
	includeInFee	Public	✓	onlyOwner
	_set_Fees	External	✓	onlyOwner
	Wallet_Update_Dev	Public	✓	onlyOwner

	set_Swap_And_Liquify_Enabled	Public	✓	onlyOwner
	set_Number_Of_Transactions_Before_Liquify_Trigger	Public	✓	onlyOwner
	blacklist_Add_Wallets	External	✓	onlyOwner
	blacklist_Remove_Wallets	External	✓	onlyOwner
	blacklist_Switch	Public	✓	onlyOwner
	set_Transfers_Without_Fees	External	✓	onlyOwner
	set_Max_Transaction_Percent	External	✓	onlyOwner
	set_Max_Wallet_Percent	External	✓	onlyOwner
	removeAllFee	Private	✓	
	restoreAllFee	Private	✓	
	_approve	Private	✓	
	_transfer	Private	✓	
	sendToWallet	Private	✓	
	swapAndLiquify	Private	✓	lockTheSwap
	process_Tokens_Now	Public	✓	onlyOwner
	swapTokensForBNB	Private	✓	
	remove_Random_Tokens	Public	✓	onlyOwner
	set_New_Router_and_Make_Pair	Public	✓	onlyOwner
	set_New_Router_Address	Public	✓	onlyOwner
	set_New_Pair_Address	Public	✓	onlyOwner
	_tokenTransfer	Private	✓	
	_transferTokens	Private	✓	
	_getValues	Private		

Inheritance Graph



Flow Graph



Summary

SafeBit contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. There are some functions that can be abused by the owner like stopping transactions, manipulating the fees, and massively blacklist addresses. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats.

Update 01/01/2024

The team has renounced the ownership.

https://bscscan.com/tx/0xa00afc62a0b8152c72984a8899754552ffa36b15748bf689a1733c395f242e06

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io