Cyberscope

Audit Report

AgaTech

September 2023

Network	BSC
Address	0xb427e47e8fdd678278d2a91eeac014ffcddaf029
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
●	RSRS	Redundant SafeMath Require Statement	Unresolved
●	RNRM	Redundant No Reentrant Modifier	Unresolved
●	RTC	Redundant Type Casting	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L16	Validate Variable Setters	Unresolved

Table of Contents

Review

Contract Name	Agatech
Compiler Version	v0.8.18+commit.87f61d96
Optimization	200 runs
Explorer	https://bscscan.com/address/0xb427e47e8fdd678278d2a91eeac014ffcddaf029
Address	0xb427e47e8fdd678278d2a91eeac014ffcddaf029
Network	BSC
Symbol	AGATA
Decimals	18
Total Supply	10,000,000

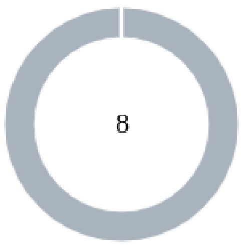
Audit Updates

Initial Audit	27 Sep 2023

Source Files

Filename	SHA256
Agatech.sol	db9da3275be9e3bc0703d1563a3dc84d6d75919085b4ec444693a2f47cb2ea41

Findings Breakdown



●	Critical	0
●	Medium	0
●	Minor / Informative	8

Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	0	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	8	0	0	0

RSRS - Redundant SafeMath Require Statement

Criticality	Minor / Informative
Location	Agatech.sol#L
Status	Unresolved

Description

The contract utilizes a `require` statement within the `add` function aiming to prevent overflow errors. This function is designed based on the SafeMath library's principles. In Solidity version 0.8.0 and later, arithmetic operations revert on overflow and underflow, making the overflow check within the function redundant. This redundancy could lead to extra gas costs and increased complexity without providing additional security.

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

Recommendation

It is recommended to remove the `require` statement from the `add` function since the contract is using a Solidity pragma version equal to or greater than 0.8.0. By doing so, the contract will leverage the built-in overflow and underflow checks provided by the Solidity language itself, simplifying the code and reducing gas consumption. This change will uphold the contract's integrity in handling arithmetic operations while optimizing for efficiency and cost-effectiveness.

RNRM - Redundant No Reentrant Modifier

Criticality	Minor / Informative
Location	Agatech.sol#L143,161
Status	Unresolved

Description

The contract uses the `nonReentrant` modifier to the `transfer` and `transferFrom` functions, which suggests an intention to prevent potential reentrancy attacks. However, neither of these functions deals with the transfer of the native token or any other value. As such, the risk of reentrancy attacks in these specific functions is minimal to non-existent.

```
function transfer(address recipient, uint256 amount) external override
nonReentrant returns (bool) {
    _transfer(msg.sender, recipient, amount);
    return true;
}

function transferFrom(address sender, address recipient, uint256 amount)
external override nonReentrant returns (bool) {
    _transfer(sender, recipient, amount);
    uint256 currentAllowance = _allowances[sender][msg.sender];
    require(currentAllowance >= amount, "Transfer amount exceeds
allowance");
    _approve(sender, msg.sender, currentAllowance.sub(amount));
    return true;
}
```

Recommendation

To address this finding and enhance code simplicity and clarity, it is recommended to remove the unnecessary `nonReentrant` modifier from the `transfer` and `transferFrom` functions. By removing the modifier, the code becomes more streamlined and easier to comprehend, reducing the gas consuption.

RTC - Redundant Type Casting

Criticality	Minor / Informative
Location	Agatech.sol#L62
Status	Unresolved

Description

The contract is initializing the `_totalSupply` variable using a calculation that includes `decimals`, which is redundantly cast to `uint256`. Given that `decimals` is already of type `uint8`, this explicit type casting is unnecessary and could lead to confusion. Additionally, the value of decimals is a constant 18, making the type casting not only redundant but also potentially gas inefficient.

```
uint256 private _totalSupply = 10000000 * (10 ** uint256(decimals));
```

Recommendation

It is recommended to remove the redundant type casting of `uint256` in `decimals` varriable. Since `decimals` is already of type `uint8`, the explicit type casting serves no functional purpose and could lead to misunderstandings about the code's intent.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	Agatech.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.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	Agatech.sol#L72
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
AgatechMultisig
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	Agatech.sol#L62
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.

```
uint256 private _totalSupply = 10000000 * (10 ** uint256(decimals))
```

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	Agatech.sol#L65
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.

```
address public AgatechMultisig
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	Agatech.sol#L190
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.

```
(bool success,) = target.call{value: value}(data)
```

Recommendation

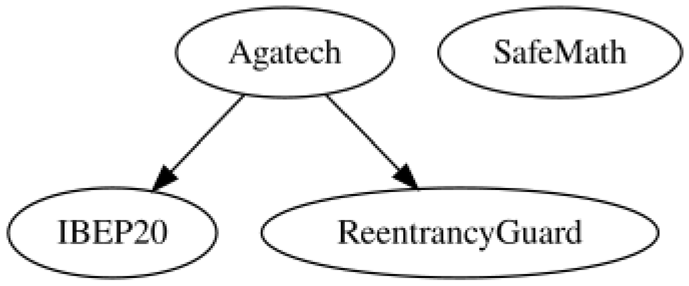
By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

Functions Analysis

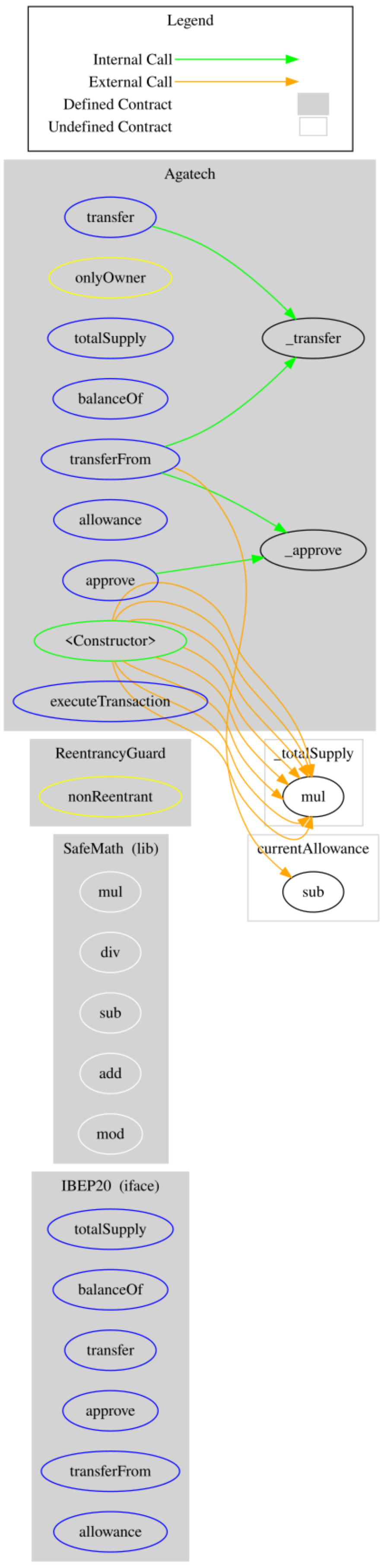
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IBEP20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	approve	External	✓	-
	transferFrom	External	✓	-
	allowance	External		-
SafeMath	Library			
	mul	Internal		
	div	Internal		
	sub	Internal		
	add	Internal		
	mod	Internal		
ReentrancyGuard	Implementation			

Agatech	Implementation	IBEP20, ReentrancyGuard		
		Public	✓	-
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	nonReentrant
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	nonReentrant
	_transfer	Internal	✓	
	_approve	Internal	✓	
	executeTransaction	External	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Agatech
transfer
onlyOwner
totalSupply
balanceOf
transferFrom
allowance
approve
<Constructor>
executeTransaction
_transfer
_approve
ReentrancyGuard
nonReentrant
_totalSupply
mul
currentAllowance
sub
SafeMath (lib)
mul
div
sub
add
mod
IBEP20 (iface)
totalSupply
balanceOf
transfer
approve
transferFrom
allowance

Summary

AgaTech contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. AgaTech is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues. The contract Owner can access some admin functions that can not be used in a malicious way to disturb the users' transactions.

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