VitalBlock

FOR

Awfos Token

WEBSITE

https://awfos.com/

TELEGRAM

https://t.me/awfostoken

SMART CONTRACT AUDIT

Twitter:
twitter.com/vital_block

Telegram:
t.me/vital_Block

Disclaimer

Vital Block Solidity reports are not, nor should be considered, an "endorsement" or "disapproval" of any project or team. These reports are not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Vital Block to perform a security review.

Vital Black Solidity Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analysed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

Vital Block Solidity Reports should not be used in any way to make decisions around investment or involvement with any project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort. Vital Block Solidity Reports represent an extensive auditing 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. Vital Block Solidity's position is that each company and individual are responsible for their own due diligence and continuous security. Vital Block Solidity'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 analyse

What is a Vital Block Audit report?

- A document describing in detail an in-depth analysis of a particular piece(s) of source code provided to Vital Block Solidity by a Client.

- An organized collection of testing results, analysis and inferences made about the structure, implementation, and overall best practices of a particular piece of source code.

- Representation that a Client of Vital Block Solidity has indeed completed a round of auditing with the intention to increase the quality of the company/ product's IT infrastructure and or source code.

Overview

Project Summary

Project Name	**AWFOS TOKEN**
Description	**It takes a community to build a project.**
Platform	**Binance Mainnet**
Mainnet Contracts:	**0x93D101BE221c4f28020b818C31212865Fb39Ec60** ***AWFOS TOKEN (AFS)***
Files:	**AWFOS.sol**

Audit Summary

Delivery Date	**June 112022**
Method of Audit	**Static Analysis**
Timeline	**Story Points 100**

Vulnerability Summary

Total Issues Found	**2**
Total Issues Resolved	**2**
Total Critical	**0**
Total High	**1**
Total Medium	**3**
Total Low	**0**
Total Informational	**1**

Executive Summary

Our Audit Methodology

- **STEP 1**
  A manual line-by-line code review to ensure the logic behind each function is safe and secured against common attack vectors.

- **STEP 2**
  Simulation of hundreds of thousands of Smart Contract Interactions on a test and Mainnet blockchain using a combination of automated test tools and manual testing to determine if any security vulnerabilities exist.

- **STEP 3**
  Consultation with the project team on the audit report pre-publication to implement recommendations and resolve any outstanding issues.

Grading

The following grading structure is used to assess the level of vulnerability found within all Smart Contracts:

THREAT LEVEL	DEFINITION
Critical	Severe vulnerabilities which compromise the entire protocol and could result in immediate data manipulation or asset loss.
High	Significant vulnerabilities which compromise the functioning of the smart contracts leading to possible data manipulation or asset loss.
Medium	Vulnerabilities which if not fixed within in a set timescale could compromise the functioning of the smart contracts leading to possible data manipulation or asset loss.
Low	Low level vulnerabilities which may or may not have an impact on the optimal performance of the Smart contract.
Informational	Issues related to coding best practice which do not have any impact on the functionality of the Smart Contracts

Description

AWFOS ($AFS) is a deflationary token designed to become scarcer over time. Eligible Holders* of $AFS will earn an 2% reward from every Buy/Transfer/Sell Transaction , which is automatically sent to your wallet.

Trading fees are 4% on buys and a 4% on sells. The fees are distributed as follows:

Buy Trading Fees 4.0% - LP | 1% - Marketing & Development | 1% - Buyback | 2% - Reward

Sell Trading Fees 4.0% - LP | 1% - Marketing & Development | 1% - Buyback | 2% - Reward

Initial supply is 5,000,000,000 tokens, distributed as follows

20% - Presale

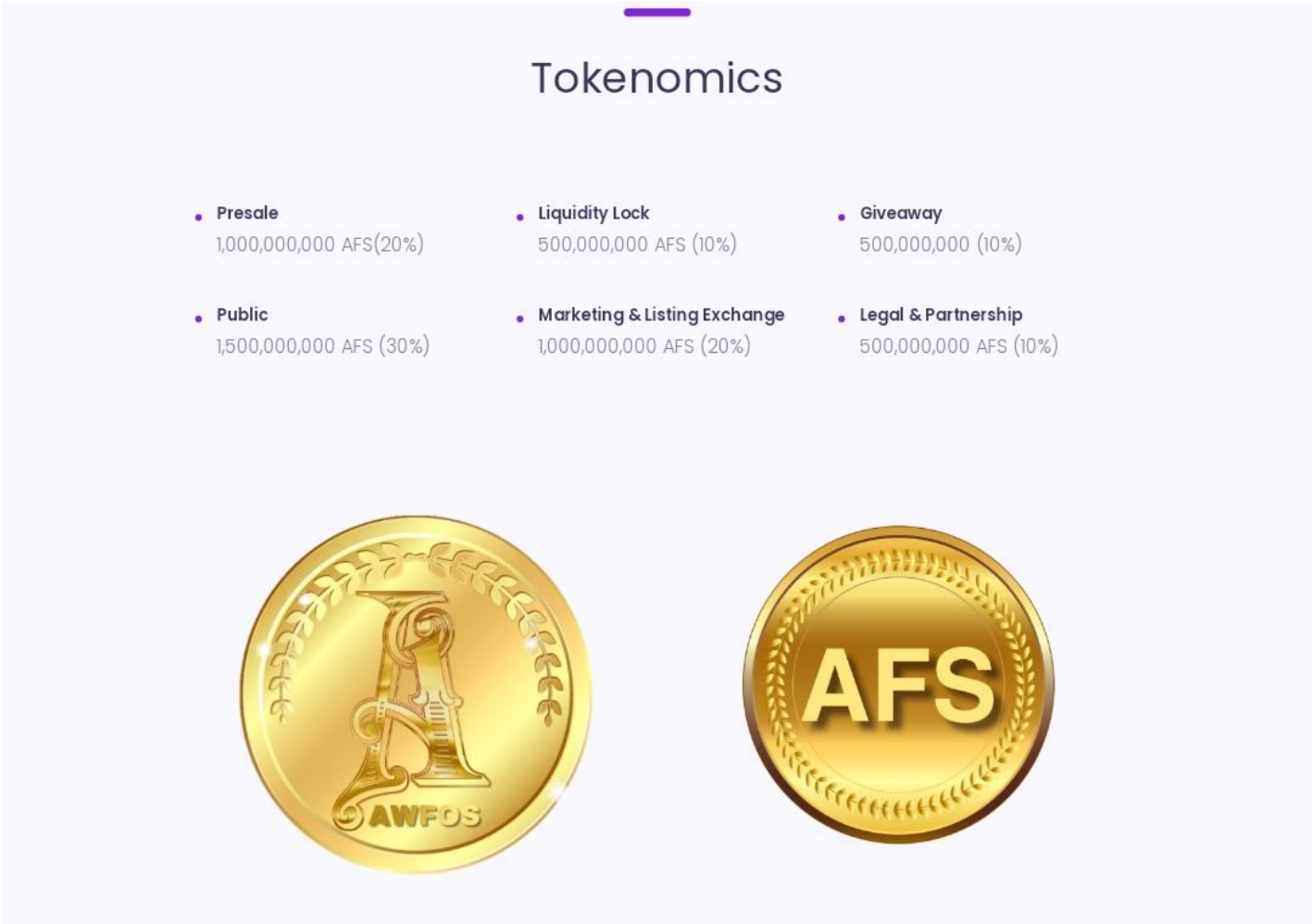
10% - Initial Liquidity

30% – Public

20% – Marketing & Listing

10% - Giveaway

10% - Legal & Partnership



Vulnerability 1: Total Supply cant exceed MAX_SUPPLY

Threat level: Medium

Description:

The contract checks whether total Supply is Equal To MAX_SUPPLY. And there is A check to return Supply function when total supply Tends to exceeds MAX_SUPPLY.

```
857
858     function _getRValues(uint256 tAmount, uint256 tFee, uint256 tLiquidity, uint256 current
859         uint256 rAmount = tAmount.mul(currentRate);
860         uint256 rFee = tFee.mul(currentRate);
861         uint256 rLiquidity = tLiquidity.mul(currentRate);
862         uint256 rTransferAmount = rAmount.sub(rFee).sub(rLiquidity);
863         return (rAmount, rTransferAmount, rFee);
864     }
865
866     function _getRate() private view returns(uint256) {
867         (uint256 rSupply, uint256 tSupply) = _getCurrentSupply();
868         return rSupply.div(tSupply);
869     }
870
871     function _getCurrentSupply() private view returns(uint256, uint256) {
872         uint256 rSupply = _rTotal;
873         uint256 tSupply = _tTotal;
874         for (uint256 i = 0; i < _excluded.length; i++) {
875             if (_rOwned[_excluded[i]] > rSupply || _tOwned[_excluded[i]] > tSupply) return
876             rSupply = rSupply.sub(_rOwned[_excluded[i]]);
877             tSupply = tSupply.sub(_tOwned[_excluded[i]]);
878         }
879         if (rSupply < _rTotal.div(_tTotal)) return (_rTotal, _tTotal);
880         return (rSupply, tSupply);
881     }
```

Vulnerability 1: Owner Can Remove Fee up to 0

Threat level: Informational

Vulnerability 1: Gas optimisation

Threat level 0: Informational

Description: Does not seem like a honeypot.
This can always change! Do your own due diligence.
INFO! Owner can change trading tax fee up to 0 which is a good call on the Smart Contract. Removal fee is private and calculate function

AWFOS (AFS)

Max TX: 15000000 AFS (~0 BNB)
Gas used for Buying: 216,472
Gas used for Selling: 174,247

Recommendation:

The contract can be modified so that it can be done via a single call to save gas.

```
895     }
896
897     function calculateLiquidityFee(uint256 _amount) private view returns (uint256) {
898         return _amount.mul(_liquidityFee).div(
899             10**3
900         );
901     }
902
903     function removeAllFee() private {
904         if(_taxFee == 0 && _liquidityFee == 0) return;
905
906         _previousTaxFee = _taxFee;
907         _previousLiquidityFee = _liquidityFee;
908         previousBuybackFee = buybackFee;
909         previousMarketingFee = marketingFee;
910
911         _taxFee = 0;
912         _liquidityFee = 0;
913         buybackFee = 0;
914         marketingFee = 0;
915     }
916
917     function restoreAllFee() private {
918         _taxFee = _previousTaxFee;
919         _liquidityFee = _previousLiquidityFee;
920         buybackFee = previousBuybackFee;
921         marketingFee = previousMarketingFee;
922     }
```

Issues Checking Status

	Issue description	Checking status
1.	**Compiler errors.**	Passed
2.	**Race conditions and Reentrancy. Cross-function race conditions.**	Passed
3.	**Possible delays in data delivery.**	Passed
4.	**Oracle calls.**	Passed
5.	**Front running.**	Passed
6.	**Timestamp dependence.**	Passed
7.	**Integer Overflow and Underflow.**	Passed
8.	**DoS with Revert.**	Passed
9.	**DoS with block gas limit.**	Passed
10.	**Methods execution permissions.**	Passed
11.	**Economy model of the contract.**	Passed
12.	**The impact of the exchange rate on the logic.**	Passed
13.	**Private user data leaks.**	Passed
14.	**Malicious Event log.**	Passed
15.	**Scoping and Declarations.**	Passed
16.	**Uninitialized storage pointers.**	Passed
17.	**Arithmetic accuracy.**	Passed
18.	**Design Logic.**	Passed
19.	**Cross-function race conditions.**	Passed
20.	**Safe Open Zeppelin contracts implementation and usage.**	Passed
21.	**Fallback function security.**	Passed

Conclusion

During the Vital block Audit process, the Awfos contract was analysed by manual review and automated testing. All issues identified pre-launch were resolved before deployment to mainnet. By submitting the contract for audit pre-launch, the team have displayed a strong commitment to security.

Whilst there are no obvious vulnerabilities or security risks identified within the main net contract, it is beyond the scope of this Vital Block Audit to comment upon any risks associated with tokenomics, adoption or platform longevity. Before placing funds in any defi protocol Vital Block encourages potential investors to exercise due diligence and research all projects thoroughly to assess plans for ongoing development and financial sustainability.

Appendix

Finding Categories

Gas Optimization
Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations
Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue
Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Control Flow
Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code
Volatile Code findings refer to segments of code that behave unexpectely on certain edge cases that may result in avulnerability.

Data Flow
Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a structassignment operation affecting an in-memory struct rather than an instorage one.

Language Specific
Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.

Coding Style
Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Appendix

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as constant contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.

VitalBlock

WWW.VITALBLOCK.XYZ

Decentralized Smart Contract Auditing Firm.