Audit Report

Ash Token

March 2024

Repository	https://github.com/TheAshDAO/AshToken
Repository	c0fc2bad269879524ab14056ae13f668a0b65257
Audited by	© cyberscope

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	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
●	DDP	Decimal Division Precision	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	MTEE	Missing Transfer Event Emission	Unresolved
●	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	RFC	Redundant Fee Calculations	Unresolved
●	RRS	Redundant Require Statement	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	RSD	Redundant Swap Duplication	Unresolved
●	OCTD	Transfers Contract's Tokens	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L13	Divide before Multiply Operation	Unresolved

●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Review

Repository	https://github.com/TheAshDAO/AshToken
Commit	c0fc2bad269879524ab14056ae13f668a0b65257
Testing Deploy	https://testnet.bscscan.com/address/0x0968eb1f7d6e599a5376b3b3fd83246ee688bc1d
Badge Eligibility	Yes

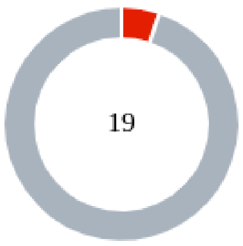
Audit Updates

Initial Audit	21 Mar 2024

Source Files

Filename	SHA256
AshToken.sol	31712df46f888ab99da94e2422519e141b9c3a6464aa575d7c1c5a25dbc0cca1

Findings Breakdown



●	Critical	1
●	Medium	0
●	Minor / Informative	18

Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	1	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	18	0	0	0

ST - Stops Transactions

Criticality	Critical
Status	Unresolved

Description

The contract owner has the authority to stop transactions, as described in detail in the `PTRP` section. As a result, the contract might operate as a honeypot.

Recommendation

It is advised to implement checks that limit the contract's ability to unilaterally stop transactions, as outlined in `PTRP` section. This precaution helps mitigate the risk of the contract being used as a honeypot, ensuring a more secure and trustworthy environment for users.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	AshToken.sol#L631
Status	Unresolved

Description

Division of decimal (fixed point) numbers can result in rounding errors due to the way that division is implemented in Solidity. Thus, it may produce issues with precise calculations with decimal numbers.

Solidity represents decimal numbers as integers, with the decimal point implied by the number of decimal places specified in the type (e.g. decimal with 18 decimal places). When a division is performed with decimal numbers, the result is also represented as an integer, with the decimal point implied by the number of decimal places in the type. This can lead to rounding errors, as the result may not be able to be accurately represented as an integer with the specified number of decimal places.

Hence, the splitted shares will not have the exact precision and some funds may not be calculated as expected.

```
uint256 tDao = transFee.mul(daoFundTax).div(1000);
uint256 tMarketing = transFee.mul(marketingTax).div(1000);
uint256 tLiquidity = transFee.mul(liquidityTax).div(1000);
uint256 tFee = transFee.mul(reflectionsTax).div(1000);
uint256 tBurning = transFee.mul(burningTax).div(1000);
```

Recommendation

The team is advised to take into consideration the rounding results that are produced from the solidity calculations. The contract could calculate the subtraction of the divided funds in the last calculation in order to avoid the division rounding issue.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	AshToken.sol#L354,368
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.

```
dexRouter
lpPair
```

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.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	AshToken.sol#L485
Status	Unresolved

Description

The contract operates under the assumption that liquidity is consistently provided to the pair between the contract's token and the native currency. However, there is a possibility that liquidity is provided to a different pair. This inadequacy in liquidity provision in the main pair could expose the contract to risks. Specifically, during eligible transactions, where the contract attempts to swap tokens with the main pair, a failure may occur if liquidity has been added to a pair other than the primary one. Consequently, transactions triggering the swap functionality will result in a revert.

```
address[] memory path = new address[](2);
path[0] = address(this);
path[1] = IDexRouter(dexRouter).WETH()
_approve(address(this), address(dexRouter), tokenAmount)
IDexRouter(dexRouter)
    .swapExactTokensForETHSupportingFeeOnTransferTokens(
        tokenAmount,
        0,
        path,
        receiver,
        block.timestamp
    );
```

Recommendation

The team is advised to implement a runtime mechanism to check if the pair has adequate liquidity provisions. This feature allows the contract to omit token swaps if the pair does not have adequate liquidity provisions, significantly minimizing the risk of potential failures.

Furthermore, the team could ensure the contract has the capability to switch its active pair in case liquidity is added to another pair.

Additionally, the contract could be designed to tolerate potential reverts from the swap functionality, especially when it is a part of the main transfer flow. This can be achieved by executing the contract's token swaps in a non-reversible manner, thereby ensuring a more resilient and predictable operation.

MTEE - Missing Transfer Event Emission

Criticality	Minor / Informative
Location	386AshToken.sol#L386
Status	Unresolved

Description

The contract is a missing transfer event emission when fees are transferred to the contract address as part of the transfer process. This omission can lead to a lack of visibility into fee transactions and hinder the ability of decentralized applications (DApps) like blockchain explorers to accurately track and analyze these transactions.

```
_rOwned[msg.sender] = _rTotal;
```

Recommendation

To address this issue, it is recommended to emit a transfer event after transferring the taxed amount to the contract address. The event should include relevant information such as the sender, recipient (contract address), and the amount transferred.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	AshToken.sol#L496
Status	Unresolved

Description

The contract sends funds to a `DAO_ADDRESS` and `MARKETING_ADDRESS` addresses as part of the transfer flow. These addresses can either be a wallet address or a contract. If the address belongs to a contract then it may revert from incoming payment. As a result, the error will propagate to the token's contract and revert the transfer.

```
IDexRouter(dexRouter)
    .swapExactTokensForETHSupportingFeeOnTransferTokens(
        tokenAmount,
        0,
        path,
        receiver,
        block.timestamp
    );
```

Recommendation

The contract should tolerate the potential revert from the underlying contracts when the interaction is part of the main transfer flow. This could be achieved by not allowing set contract addresses or by sending the funds in a non-revertable way.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	AshToken.sol#L455,463,501
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variableS `daoThreshold` and `marketingThreshold` set 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.

```
if (balanceOf(DAO_ADDRESS) + tDao > daoThreshold) {
     _takeFee(sender, tDao, address(this));
     swapTokensForBnb(tDao, DAO_ADDRESS);
 } else {
     _takeFee(sender, tDao, DAO_ADDRESS);

 if (
     balanceOf(MARKETING_ADDRESS) + tMarketing >
marketingThreshold
 ) {
     _takeFee(sender, tMarketing, address(this));
     swapTokensForBnb(tMarketing, MARKETING_ADDRESS);

    ...
    function swapAndEvolve() public onlyOwner lockTheSwap {
        // split the contract balance into halves
        uint256 contractAshBalance = balanceOf(address(this));
        ...

        // swap ASH for BNB
        swapTokensForBnb(half, address(this));
    ...
    }
```

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.

RFC - Redundant Fee Calculations

Criticality	Minor / Informative
Location	AshToken.sol#L455
Status	Unresolved

Description

The contract is calling the `_takeFee` function multiple times within conditional (if-else) blocks for different purposes (DAO, marketing, liquidity). This approach, while functional, leads to unnecessary repetition of the `_takeFee` call, which could be streamlined. Since `_takeFee` is invoked in both the if and else branches of the conditionals, and its execution is guaranteed regardless of the condition's outcome, this redundancy could be optimized. The current implementation may not only increase gas costs but also complicates the contract's logic unnecessarily.

```
if (balanceOf(DAO_ADDRESS) + tDao > daoThreshold) {
    _takeFee(sender, tDao, address(this));
    swapTokensForBnb(tDao, DAO_ADDRESS);
} else {
    _takeFee(sender, tDao, DAO_ADDRESS);

if (
    balanceOf(MARKETING_ADDRESS) + tMarketing > marketingThreshold
) {
    _takeFee(sender, tMarketing, address(this));
    swapTokensForBnb(tMarketing, MARKETING_ADDRESS);
} else {
    _takeFee(sender, tMarketing, MARKETING_ADDRESS);

_takeFee(sender, tLiquidity, address(this));
_takeBurn(sender, tBurning);
```

Recommendation

It is recommended to refactor the contract logic to call `_takeFee` outside of the conditional blocks when its execution is required regardless of the conditions being checked. This change would simplify the contract's logic, potentially reduce gas costs, and improve code maintainability. Consolidating the `_takeFee` calls into a single location, when possible, will make the contract easier to understand and audit, reducing the risk of errors in fee handling logic.

RRS - Redundant Require Statement

Criticality	Minor / Informative
Location	AshToken.sol#L90
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.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	AshToken.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 in cases where the explanatory error message is not used.

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

Recommendation

The team is advised to remove the SafeMath library in cases where the revert error message is not used. 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.

RSD - Redundant Swap Duplication

Criticality	Minor / Informative
Location	AshToken.sol#L457,466
Status	Unresolved

Description

The contract contains multiple swap methods that individually perform token swaps and transfer promotional amounts to specific addresses and features. This redundant duplication of code introduces unnecessary complexity and increases dramatically the gas consumption. By consolidating these operations into a single swap method, the contract can achieve better code readability, reduce gas costs, and improve overall efficiency.

```
   swapTokensForBnb(tDao, DAO_ADDRESS);
} else {
    _takeFee(sender, tDao, DAO_ADDRESS);

if (
    balanceOf(MARKETING_ADDRESS) + tMarketing >
marketingThreshold
) {
    _takeFee(sender, tMarketing, address(this));
    swapTokensForBnb(tMarketing, MARKETING_ADDRESS);
```

Recommendation

A more optimized approach could be adopted to perform the token swap operation once for the total amount of tokens and distribute the proportional amounts to the corresponding addresses, eliminating the need for separate swaps.

OCTD - Transfers Contract's Tokens

Criticality	Minor / Informative
Location	AshToken.sol#L682
Status	Unresolved

Description

The contract owner has the authority to claim all the balance of the contract. The owner may take advantage of it by calling the `claimTokens` function.

```
    function claimStuckTokens(address _token) external onlyOwner
{
        IERC20 token = IERC20(_token);
        token.transfer(owner(),
token.balanceOf(address(this)));
    }
```

Recommendation

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	AshToken.sol#L301,306,307,308,309,310
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 public maxTax = 100
uint256 public daoFundTax = 800
uint256 public marketingTax = 135
uint256 public liquidityTax = 25
uint256 public reflectionsTax = 25
uint256 public burningTax = 15
```

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	AshToken.sol#L236,317,318,322,323,325,326,682,710,720,731,732,742,750
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 WETH() external pure returns (address);
address public DAO_ADDRESS
address public MARKETING_ADDRESS
uint256 public _rTotal = (MAX - (MAX % _tTotal))
uint256 public _tFeeTotal
mapping(address => uint256) public _rOwned
mapping(address => uint256) public _tOwned
address _token
address _address
uint256 _daoThreshold
uint256 _marketingThreshold
uint256 _transferTax
uint256 _sellTax
uint256 _buyTax
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	AshToken.sol#L629,631,632,633,634,635
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
uint256 transFee = tAmount.mul(feeAmount).div(1000)
uint256 tMarketing = transFee.mul(marketingTax).div(1000)
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	AshToken.sol#L349
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.

```
address wbnb
```

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	AshToken.sol#L383,384
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.

```
DAO_ADDRESS = _daoAddress
MARKETING_ADDRESS = _marketingAddress
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	AshToken.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 ^0.8.20;
```

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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	AshToken.sol#L684
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
token.transfer(owner(), token.balanceOf(address(this)))
```

Recommendation

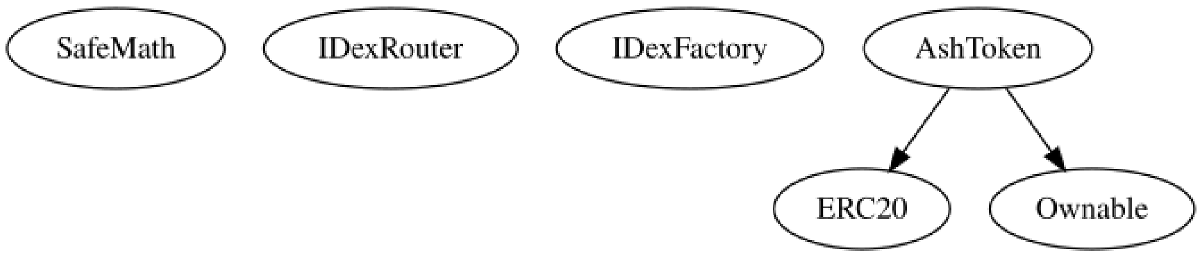
The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

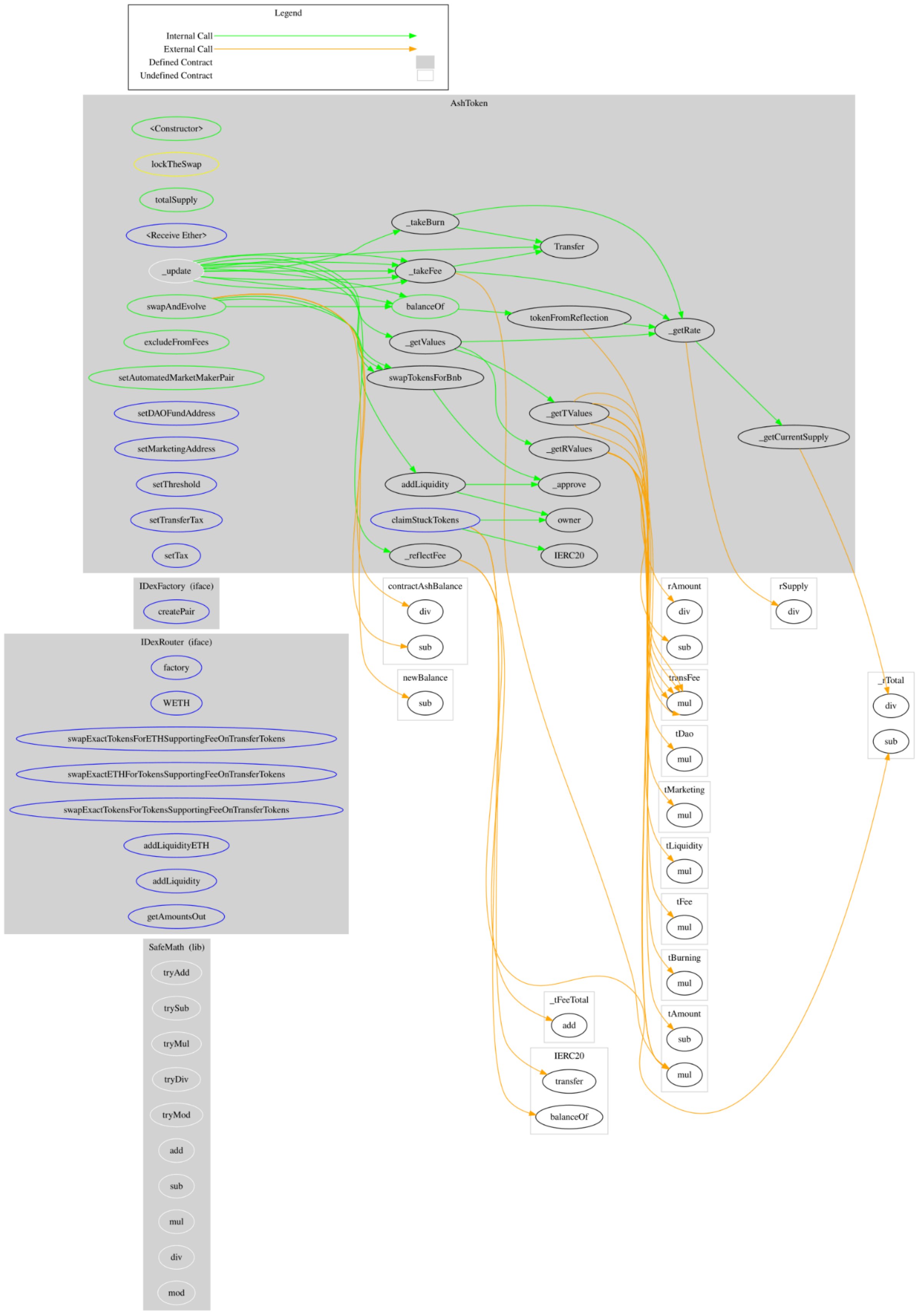
Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
AshToken	Implementation	ERC20, Ownable		
		Public	✓	ERC20 Ownable
	totalSupply	Public		-
	balanceOf	Public		-
		External	Payable	-
	tokenFromReflection	Public		-
	_update	Internal	✓	
	swapTokensForBnb	Private	✓	lockTheSwap
	swapAndEvolve	Public	✓	onlyOwner lockTheSwap
	addLiquidity	Private	✓	
	_takeFee	Private	✓	
	_takeBurn	Private	✓	
	_reflectFee	Private	✓	
	_getValues	Private		
	_getTValues	Private		
	_getRValues	Private		
	_getRate	Private		
	_getCurrentSupply	Private		

	claimStuckTokens	External	✓	onlyOwner
	excludeFromFees	Public	✓	onlyOwner
	setAutomatedMarketMakerPair	Public	✓	onlyOwner
	setDAOFundAddress	External	✓	onlyOwner
	setMarketingAddress	External	✓	onlyOwner
	setThreshold	External	✓	onlyOwner
	setTransferTax	External	✓	onlyOwner
	setTax	External	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
AshToken
<Constructor>
lockTheSwap
totalSupply
<Receive Ether>
_update
swapAndEvolve
excludeFromFees
setAutomatedMarketMakerPair
setDAOFundAddress
setMarketingAddress
setThreshold
setTransferTax
setTax
_takeBurn
Transfer
_takeFee
balanceOf
tokenFromReflection
_getRate
_getValues
swapTokensForBnb
_getTValues
_getRValues
_getCurrentSupply
addLiquidity
_approve
claimStuckTokens
owner
_reflectFee
IERC20
IDexFactory (iface)
createPair
IDexRouter (iface)
factory
WETH
swapExactTokensForETHSupportingFeeOnTransferTokens
swapExactETHForTokensSupportingFeeOnTransferTokens
swapExactTokensForTokensSupportingFeeOnTransferTokens
addLiquidityETH
addLiquidity
getAmountsOut
SafeMath (lib)
tryAdd
trySub
tryMul
tryDiv
tryMod
add
sub
mul
div
mod
contractAshBalance
div
sub
newBalance
sub
rAmount
div
sub
transFee
mul
tDao
mul
tMarketing
mul
tLiquidity
mul
tFee
mul
tBurning
mul
tAmount
sub
mul
rSupply
div
_rTotal
div
sub
_tFeeTotal
add
IERC20
transfer
balanceOf

Summary

Ash Token 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 stop transactions. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats. There is also a limit of max 10% fees.

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