Cyberscope

Audit Report
Wednesday NFT

February 2023

Type	BEP20
Network	BSC
Address	0x9f55c34dc4b00659d70bea02769bae51b0c341fc
Audited by	© cyberscope

Table of Contents

Review

Contract Name	ADDAMS
Compiler Version	v0.8.4+commit.c7e474f2
Optimization	99999 runs
Explorer	https://bscscan.com/address/0x9f55c34dc4b00659d70bea02769bae51b0c341fc
Address	0x9f55c34dc4b00659d70bea02769bae51b0c341fc
Network	BSC
Symbol	ADDAMS
Decimals	18
Total Supply	1,000,000,000

Audit Updates

Initial Audit	26 Feb 2023 https://github.com/cyberscope-io/audits/tree/main/addams/v1/audit.pdf
Corrected Phase 2	28 Feb 2023

Source Files

Filename	SHA256
@openzeppelin/contracts/access/AccessControl.sol	86908de632a9fbffc04a94fa27bd320c304a47072a85de02293e08f1724934fb
@openzeppelin/contracts/access/IAccessControl.sol	d03c1257f2094da6c86efa7aa09c1c07ebd33dd31046480c5097bc2542140e45
@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol	af5c8a77965cc82c33b7ff844deb9826166689e55dc037a7f2f790d057811990
@openzeppelin/contracts/token/ERC20/IERC20.sol	94f23e4af51a18c2269b355b8c7cf4db8003d075c9c541019eb8dcf4122864d5
@openzeppelin/contracts/utils/Context.sol	1458c260d010a08e4c20a4a517882259a23a4baa0b5bd9add9fb6d6a1549814a
@openzeppelin/contracts/utils/introspection/ERC165.sol	8806a632d7b656cadb8133ff8f2acae4405b3a64d8709d93b0fa6a216a8a6154
@openzeppelin/contracts/utils/introspection/IERC165.sol	701e025d13ec6be09ae892eb029cd83b3064325801d73654847a5fb11c58b1e5
@openzeppelin/contracts/utils/math/Math.sol	8059d642ec219d0b9b62fbc76912079529cf494cac988abe5e371f1168b29b0f
@openzeppelin/contracts/utils/math/SafeMath.sol	0dc33698a1661b22981abad8e5c6f5ebca0dfe5ec14916369a2935d888ff257a
@openzeppelin/contracts/utils/Strings.sol	f81f11dca62dcd3e0895e680559676f4ba4f2e12a36bb0291d7ecbb6b983141f
@uniswap/v2-core/contracts/interfaces/IUniswapV2Factory.sol	51d056199e3f5e41cb1a9f11ce581aa3e190cc982db5771ffeef8d8d1f962a0d
@uniswap/v2-periphery/contracts/interfaces/IUniswapV2Router01.sol	0439ffe0fd4a5e1f4e22d71ddbda76d63d61679947d158cba4ee0a1da60cf663
@uniswap/v2-periphery/contracts/interfaces/IUniswapV2Router02.sol	a2900701961cb0b6152fc073856b972564f7c798797a4a044e83d2ab8f0e8d38

contracts/ADDAMS.sol	0ec6b49b156004e64bb78f00db6fe572caa4bafcc97fbd1c9b523d76f8de2b65
contracts/interfaces/IDistribution.sol	9fba229a13ad042492df1d2d065146a1e26d9719bb6f7b89b758b61d12ac15a9

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	OCTD	Transfers Contract's Tokens	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	ULTW	Transfers Liquidity to Team Wallet	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	PULT	Potential Unreachable Locked Tokens	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	MVN	Misleading Variables Naming	Unresolved
●	CR	Code Repetition	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

PULT - Potential Unreachable Locked Tokens

Criticality	Medium
Status	Unresolved

Description

The contract owner has the ability to reset the `liquidityFeeAmount` and `swapFeeAmount` amounts. This amount is the tokens that have been accumulated to the contract from the corresponding fees. If these counters reset, then the swap will not take into account these tokens. As a result, they will remain in the contract forever since there is no way to withdraw them.

```
function resetLiquidityFee() external onlyRole(DEFAULT_ADMIN_ROLE) {
    liquidityFeeAmount = 0;

    emit LiquidityFeeReseted();
}

...

function resetSwapFee() external onlyRole(DEFAULT_ADMIN_ROLE) {
    swapFeeAmount = 0;

    emit SwapFeeReseted();
}
```

Recommendation

The team is advised to carefully check the business logic of the contract. The contract should not allow having accumulated unreachable tokens.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	contracts/ADDAMS.sol#L301
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `feeLimit` sets a threshold where the contract will trigger the swap functionality. If the variable is set to a big number, then the contract may 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 setLimit(uint256 _feeLimit) external onlyRole(DEFAULT_ADMIN_ROLE) {
    feeLimit = _feeLimit;

    emit FeeLimitUpdated();
}
```

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 total supply. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

MVN - Misleading Variables Naming

Criticality	Minor / Informative
Location	contracts/ADDAMS.sol#L427
Status	Unresolved

Description

Variables can have misleading names if their names do not accurately reflect the value they contain or the purpose they serve. The contract uses some variable names that are too generic or do not clearly convey the information stored in the variable. Misleading variable names can lead to confusion, making the code more difficult to read and understand.

The contract utilizes a `burnFeeBuyRate`, `burnFeeSellRate`, and `burnFeeTransferRate` that intuitively means that this fee is burned. On the contrary, this fee may be transferred to addresses. As a result, it is working as a normal tax and not as a burn tac.

```
if (burnFeeRes > 0) {
    if (burnFeeReceivers.length > 0) {
        for (uint256 i = 0; i < burnFeeReceivers.length; i++) {
            _transferAmount(_from, burnFeeReceivers[i], _calcFee(burnFeeRes,
burnFeeReceiversRate[i]));
        }
    } else {
        _transferAmount(_from, deadAddress, burnFeeRes);
    }
}
```

Recommendation

It's always a good practice for the contract to contain variable names that are specific and descriptive. The team is advised to keep in mind the readability of the code.

CR - Code Repetition

Criticality	Minor / Informative
Status	Unresolved

Description

The contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

```
function updateBurnFeeReceivers(address[] calldata _burnFeeReceivers,
uint256[] calldata _burnFeeReceiversRate)

function updateLiquidityFeeReceivers(address[] calldata
_liquidityFeeReceivers, uint256[] calldata _liquidityFeeReceiversRate)

function updateSwapFeeReceivers(address[] calldata _swapFeeReceivers,
uint256[] calldata _swapFeeReceiversRate)
```

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help to reduce the complexity and size of the contract. For instance, the contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	contracts/ADDAMS.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 on 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 unnecessarily the gas consumption.

```
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	contracts/ADDAMS.sol#L108,109
Status	Unresolved

Description

The contract is using variables that initialize them only in the constructor. The other functions are not mutating the variables. These variables are not defined as `immutable`.

```
name
symbo
```

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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/ADDAMS.sol#L24,129,133,138,142,147,157,162,170,187,196,203,209,215,222,242,250,265,275,285,301,307,327,353,379
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 constant deadAddress =
0x000000000000000000000000000000000000dEaD
address _account
address _recipient
uint256 _amount
address _owner
address _spender
address _sender
uint256 _addedValue
uint256 _subtractedValue
address _token1
IUniswapV2Router02 _router
address _lpToken
bool _lp
address _address

...
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	contracts/ADDAMS.sol#L546
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 _burn(address _account, uint256 _amount) internal {
        require(_account != address(0), "ERC20: burn from the zero address");
        require(_account != deadAddress, "ERC20: burn from the dead address");
        require(balances[_account] >= _amount, "ERC20: burn amount exceeds
balance");

        _transferAmount(_account, deadAddress, _amount);
    }
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	contracts/ADDAMS.sol#L455,482
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 liquidityFeeHalf = liquidityFeeAmount.div(2)
uint256 liquidityFeeToken1Amount = _calcFee(token1Balance,
liquidityFeeHalf.mul(10000).div(amountToSwap))
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/ADDAMS.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 ^0.8.2;
```

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	contracts/ADDAMS.sol#L198
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.

```
IERC20(_address).transfer(msg.sender, _amount)
```

Recommendation

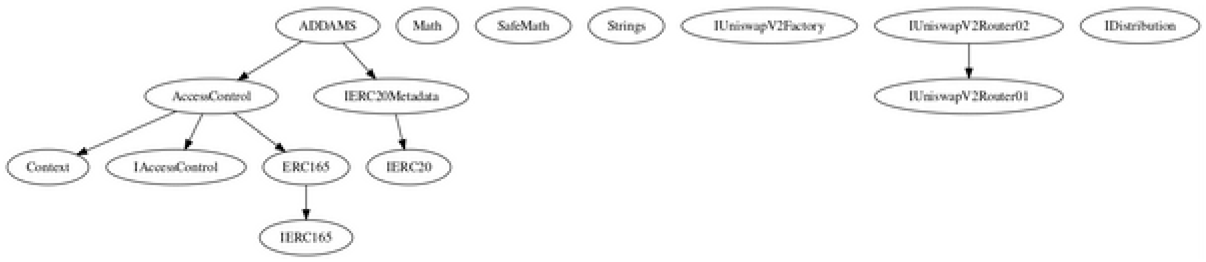
The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

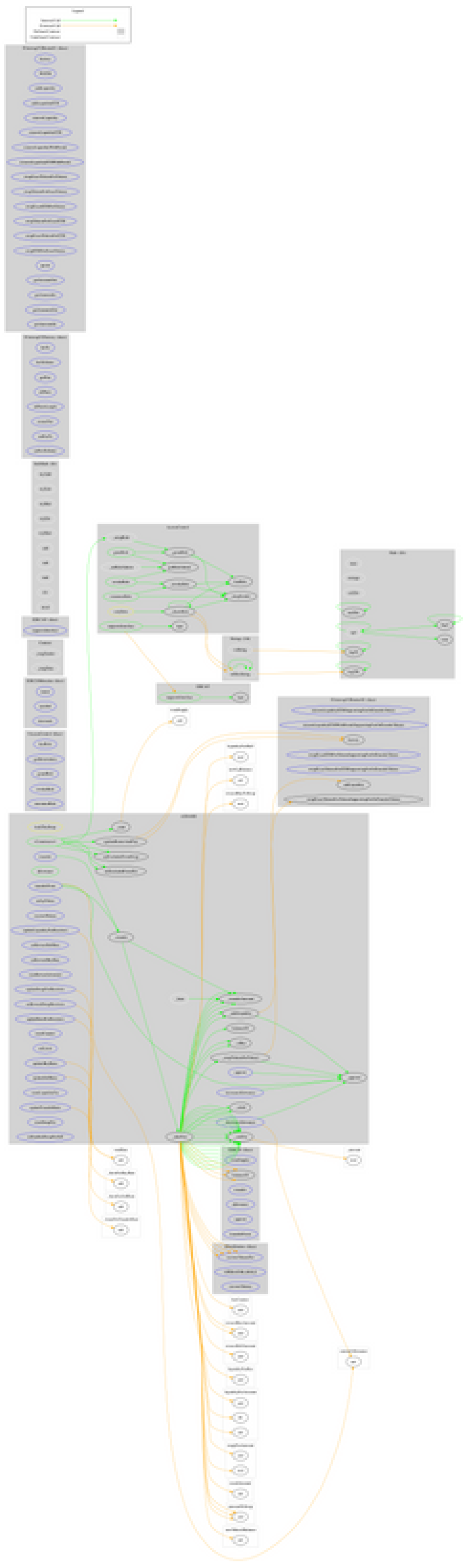
Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
ADDAMS	Implementation	IERC20Metadata, AccessControl		
		Public	✓	-
	balanceOf	Public		-
	transfer	External	✓	-
	allowance	Public		-
	approve	External	✓	-
	transferFrom	External	✓	-
	increaseAllowance	External	✓	-
	decreaseAllowance	External	✓	-
	updateRouterAndPair	Public	✓	onlyRole
	setLpToken	External	✓	onlyRole
	recoverTokens	External	✓	onlyRole
	setExcludedFromFee	Public	✓	onlyRole
	setExcludedFromSwap	Public	✓	onlyRole
	setRewardSwapReceivers	External	✓	onlyRole
	setRewardSellRate	External	✓	onlyRole
	setRewardBuyRate	External	✓	onlyRole
	resetRewardsAmount	External	✓	onlyRole
	updateBuyRates	External	✓	onlyRole
	updateSellRates	External	✓	onlyRole
	updateTransferRates	External	✓	onlyRole
	resetCounter	External	✓	onlyRole

	setLimit	External	✓	onlyRole
	updateBurnFeeReceivers	External	✓	onlyRole
	updateLiquidityFeeReceivers	External	✓	onlyRole
	resetLiquidityFee	External	✓	onlyRole
	updateSwapFeeReceivers	External	✓	onlyRole
	resetSwapFee	External	✓	onlyRole
	setEnabledSwapForSell	External	✓	onlyRole
	_transfer	Internal	✓	
	_takeFees	Internal	✓	
	_transferAmount	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_calcFee	Internal		
	_isSell	Internal		
	_isBuy	Internal		
	_swapTokensForToken1	Internal	✓	lockTheSwap
	_addLiquidity	Internal	✓	lockTheSwap

Inheritance Graph



Flow Graph



Summary

ADDAMS contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. ADDAMS is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler errors 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. There is also a limit of max 9% fees.

Disclaimer

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