Cyberscope

Audit Report

Apple Fan Metaverse

February 2023

Type	BEP20
Network	BSC
Address	0x361abc7805532735696b194a9B74a0890dC46a7B
Audited by	© cyberscope

Table of Contents

Review

Contract Name	AFM
Compiler Version	v0.8.6+commit.11564f7e
Optimization	200 runs
Explorer	https://bscscan.com/address/0x361abc7805532735696b194a9b74a0890dc46a7b
Address	0x361abc7805532735696b194a9b74a0890dc46a7b
Network	BSC
Symbol	AFM
Decimals	18
Total Supply	165,000,000

Audit Updates

Initial Audit	14 Feb 2023

Source Files

Filename	SHA256
AFM.sol	d533c5dbf14e4bf593e3f44e1e556e4af00684de95bffc585197cefe89c2d7f0

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
⚪	PTRP	Potential Transfer Revert Propagation	Unresolved
🔴	ZD	Zero Division	Unresolved
⚪	RMT	Redundant Multisign Trigger	Unresolved
⚪	MMF	Misleading Multisign Functionality	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
⚪	L08	Tautology or Contradiction	Unresolved
⚪	L09	Dead Code Elimination	Unresolved
⚪	L13	Divide before Multiply Operation	Unresolved
⚪	L15	Local Scope Variable Shadowing	Unresolved
⚪	L16	Validate Variable Setters	Unresolved

●	L17	Usage of Solidity Assembly	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	AFM.sol#L755
Status	Unresolved

Description

The contract sends funds to a `marketingWallet` as part of the transfer flow. This address 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.

```
payable(projectFeeReceiverA).sendValue((amountBNBProject / 3));
payable(projectFeeReceiverB).sendValue((amountBNBProject / 3));
payable(projectFeeReceiverC).sendValue((amountBNBProject / 3));
...
```

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 archived by not allowing set contract addresses or by sending the funds in a non-revertable way.

ZD - Zero Division

Criticality	Critical
Location	AFM.sol#L759
Status	Unresolved

Description

The contract is using variables that may be set to zero as denominators. This can lead to unpredictable and potentially harmful results, such as a transaction revert.

```
uint256 amountToBurn          = swapThreshold * burnFee / totalFee;
```

Recommendation

It is important to handle division by zero appropriately in the code to avoid unintended behavior and to ensure the reliability and safety of the contract. The contract should ensure that the divisor is always non-zero before performing a division operation. It should prevent the variables to be set to zero or should not allow executing of the corresponding statements.

RMT - Redundant Multisign Trigger

Criticality	Minor / Informative
Location	AFM.sol#L1285
Status	Unresolved

Description

The method `setMaxWalletPercent` sets the `_maxWalletToken` using a double-signature pattern. When the initial sign process is triggered, the `maxWallPercent` is not validated. Later, in the second signature step, it is validated. As a result, if the `maxWallPercent` value in the initial sign is not valid, the multi-signature mechanism will be triggered even if it will not be possible to set the value.

```
function setMaxWalletPercent(uint256 maxWallPercent) external {
    // MULTI-SIGNATURE ID
    uint256 id = 13;
    // GLOBAL REQUIREMENTS
    checkAuth(msg.sender);
    if (ZERO == multiSignatureAddress) {
        // SETTING UP MULTI-SIGNATURE
        multiSignatureTrigger(id, msg.sender);
        _tmpMaxWalletPercent = maxWallPercent;
    }
    else {
        // GLOBAL MULTI-SIGNATURE REQUIREMENTS
        multiSignatureRequirements(id, msg.sender, true);
        // LOCAL MULTI-SIGNATURE REQUIREMENTS
        require(maxWallPercent > 0 && maxWallPercent <= 100, "Invalid
maxWallPercent");
        require(_tmpMaxTxAmount == maxWallPercent, "Invalid parameters");
        // NICE JOB. YOU DID IT!
        _maxWalletToken = (totalSupply() * maxWallPercent ) / 100;

        // RESET AFTER SUCESSFULLY COMPLETING TASK
        resetMultiSignature();
    }
}
```

Recommendation

The `setMaxWalletPercent` value should be checked before the initial trigger of the multi-sign process.

MMF - Misleading Multisign Functionality

Criticality	Minor / Informative
Location	AFM.sol#L892
Status	Unresolved

Description

The contract is using a method for securing the mutate operations, this method is called multi-signature. Intuitively the multi-signature means that the change should be approved from many wallets. On the contrary, the implementation required always two wallets to approve the operation. As a result, it is a double-signature rather than multi-signature functionality.

Recommendation

The team is advised to rename the multi-signature methods to something closer to the actual functionality. For instance, double-signature.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	AFM.sol#L410,411,412,413,414,415,421,448,473,521,522,525
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 private MKT                                          =
0x9d9D23864c5601dc328d23281A38662c9d9D7422
address private TEAM                                         =
0x95B913C8F715F1451D4D47F19bd66CC9C9e1c52B
address private PROJECT_A                                    =
0x806d1eE6479a2C36Ac9d03E457BeBf8d655E2f29
address private PROJECT_B                                    =
0xA6B996b526d94908673afd323aEdf18e7A9fe4eD
address private PROJECT_C                                    =
0x8811665Fa46a4d776D283F73e1De39AB2B4e6E8a
address private PRESALE_ADDRESS                              =
0x407993575c91ce7643a4d4cCACc9A98c36eE1BBE
uint256 _totalSupply                                         = 165000000 * 10 ** _decimals
uint256 public teamFee                                        = 0
uint256 private launchedAt                                    = 0
bool public buyCooldownEnabled                                = true
uint8 public cooldownTimerInterval                            = 30
bool           inSwap
```

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	AFM.sol#L172,221,222,239,261,410,411,412,413,414,415,418,419,420,421,424,428,431,432,457,683,884,888,896,903,1001,1038,1065,1090,1158,1184,1340,1364
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 _newOwner
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);
address private MKT                                     =
0x9d9D23864c5601dc328d23281A38662c9d9D7422
address private TEAM                                    =
0x95B913C8F715F1451D4D47F19bd66CC9C9e1c52B
address private PROJECT_A                               =
0x806d1eE6479a2C36Ac9d03E457BeBf8d655E2f29
address private PROJECT_B                               =
0xA6B996b526d94908673afd323aEdf18e7A9fe4eD
address private PROJECT_C                               =
0x8811665Fa46a4d776D283F73e1De39AB2B4e6E8a
address private PRESALE_ADDRESS                         =
0x407993575c91ce7643a4d4cCACc9A98c36eE1BBE
string constant _name                                   = "Apple Fan Metaverse"
string constant _symbol                                 = "AFM"
uint8  constant _decimals                               = 18

...
```

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	AFM.sol#L401,403,439,473,525
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.

```
mapping (address => bool) private _isExcludedFromFee
address constant WBNB                                 =
0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c
uint256 constant INITIAL_TRANSFER_TAX_RATE          = 600
uint256 private launchedAt                            = 0
bool           inSwap
```

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	AFM.sol#L921,1028,1177,1298,1380
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.

```
_maxBuyAmount = amount
liquidityFee = _liquidityFee
swapThreshold = _amount
_maxWalletToken = (totalSupply() * maxWallPercent ) / 100
multiSignatureInterval = _time
```

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.

L08 - Tautology or Contradiction

Criticality	Minor / Informative
Location	AFM.sol#L797
Status	Unresolved

Description

A tautology is a logical statement that is always true, regardless of the values of its variables. A contradiction is a logical statement that is always false, regardless of the values of its variables.

Using tautologies or contradictions can lead to unintended behavior and can make the code harder to understand and maintain. It is generally considered good practice to avoid tautologies and contradictions in the code.

```
address(this).balance >= 0
```

Recommendation

The team is advised to carefully consider the logical conditions is using in the code and ensure that it is well-defined and make sense in the context of the smart contract.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	AFM.sol#L40,61,65,73,81,95,99,111,115,127,708
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) {
        // This method relies on extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

        uint256 size;
        // solhint-disable-next-line no-inline-assembly
        assembly {
            size := extcodesize(account)
        }
        return size > 0;
    }

...
```

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	AFM.sol#L907,1163
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.

```
require(amount > (_totalSupply / 10000) && amount < (_totalSupply / 100 * 2),
"Invalid amount. Must be reasonable.")
```

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.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	AFM.sol#L884
Status	Unresolved

Description

Local scope variable shadowing occurs when a local variable with the same name as a variable in an outer scope is declared within a function or code block. When this happens, the local variable "shadows" the outer variable, meaning that it takes precedence over the outer variable within the scope in which it is declared.

```
address _msgSender
```

Recommendation

It's important to be aware of shadowing when working with local variables, as it can lead to confusion and unintended consequences if not used correctly. It's generally a good idea to choose unique names for local variables to avoid shadowing outer variables and causing confusion.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	AFM.sol#L985,1128,1348
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.

```
_tmpTimeLockAddress = holder
_tmpBurnReceiver      = _burnFeeReceiver
_tmpWithdrawTokenAddr = _tokenAddress
```

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	AFM.sol#L47,140
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	AFM.sol#L1
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 ^0.8.6;
```

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	AFM.sol#L1358
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.

```
ERC20token.transfer(multiSignatureAddress, balance)
```

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**
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	_verifyCallResult	Private		

Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	setOwner	Internal	✓	
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-
IUniswapV2Pair	Interface			
	name	External		-
	symbol	External		-
	decimals	External		-
	totalSupply	External		-
	balanceOf	External		-
	allowance	External		-
	approve	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	DOMAIN_SEPARATOR	External		-
	PERMIT_TYPEHASH	External		-
	nonces	External		-

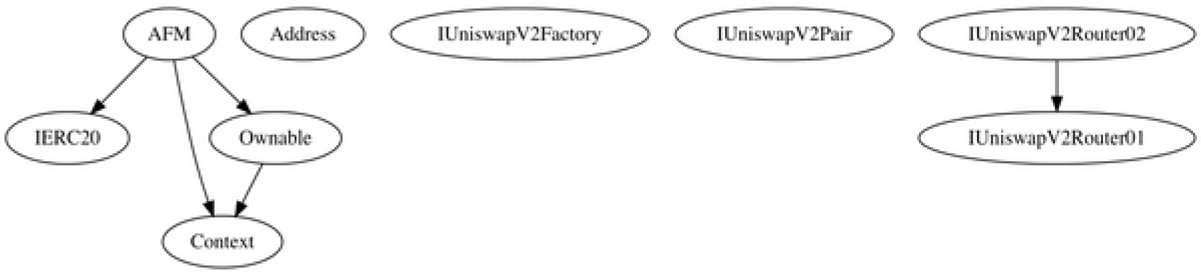
	permit	External	✓	-
	MINIMUM_LIQUIDITY	External		-
	factory	External		-
	token0	External		-
	token1	External		-
	getReserves	External		-
	price0CumulativeLast	External		-
	price1CumulativeLast	External		-
	kLast	External		-
	burn	External	✓	-
	swap	External	✓	-
	skim	External	✓	-
	sync	External	✓	-
	initialize	External	✓	-
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-
	swapTokensForExactETH	External	✓	-
	swapExactTokensForETH	External	✓	-

	swapETHForExactTokens	External	Payable	-
	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-
	getAmountsOut	External		-
	getAmountsIn	External		-
IUniswapV2Router02	Interface	IUniswapV2Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
AFM	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	✓	-

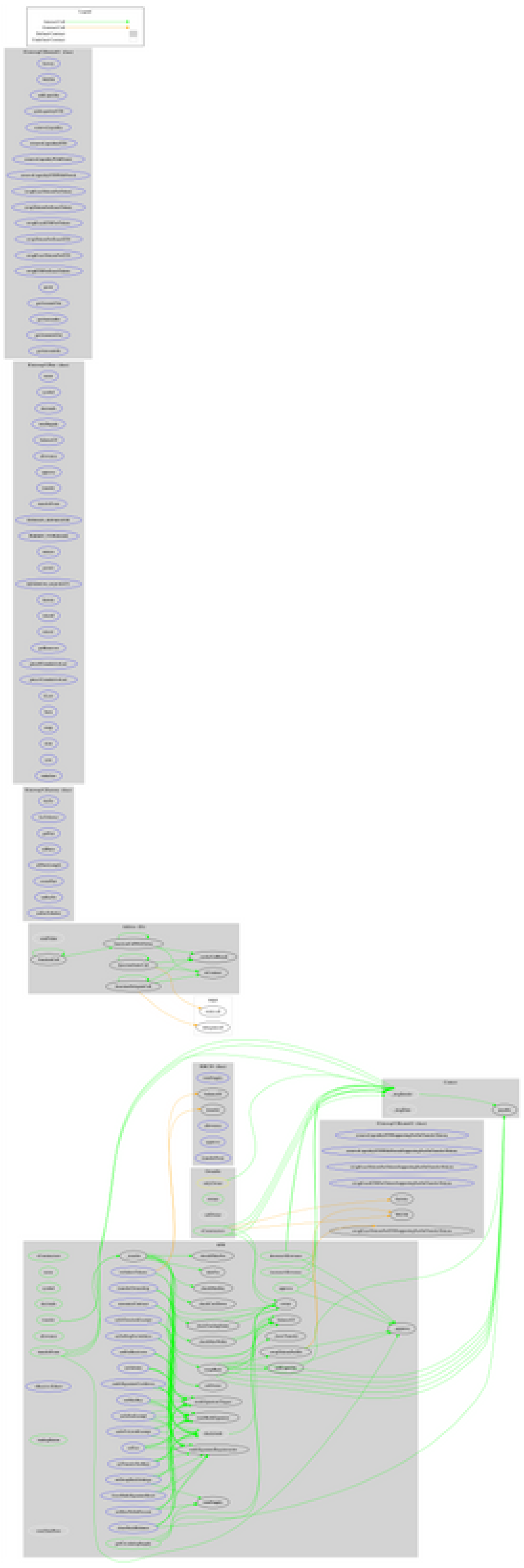
	decreaseAllowance	Public	✓	-
		External	Payable	-
	_approve	Private	✓	
	_transfer	Private	✓	
	checkCoolDown	Internal	✓	
	checkTradingStatus	Internal		
	tradingStatus	Public	✓	onlyOwner
	checkMaxBuy	Internal		
	checkMaxWallet	Internal		
	resetTotalFees	Internal	✓	
	takeFee	Internal	✓	
	shouldTakeFee	Internal		
	_basicTransfer	Internal	✓	
	swapBack	Private	✓	
	swapTokensForEth	Private	✓	
	addLiquidity	Private	✓	
	resetMultiSignature	Internal	✓	
	checkAuth	Internal		
	multiSignatureRequirements	Internal		
	multiSignatureTrigger	Internal	✓	
	setMaxBuy	External	✓	-
	setIsFeeExempt	External	✓	-
	setIsTxLimitExempt	External	✓	-
	setIsTimelockExempt	External	✓	-
	setFees	External	✓	-
	setTransferTaxRate	External	✓	-
	setSellingFeeAddress	External	✓	-
	setFeeReceivers	External	✓	-
	setSwapBackSettings	External	✓	-

	setAdmins	External	✓	-
	renounceContract	External	✓	-
	transferOwnership	External	✓	-
	setMaxWalletPercent	External	✓	-
	forceMultiSignatureReset	External	✓	-
	clearStuckBalance	External	✓	-
	withdrawTokens	External	✓	-
	multiSignatureCooldown	External	✓	-
	getCirculatingSupply	Public		-

Inheritance Graph



Flow Graph



Summary

Apple Fan Metaverse 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. There is also a limit of max 25% 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