Cyberscope

Audit Report

MetaGoldFinancial

January 2023

Type	BEP20
Network	BSC
Address	0x1f77f06333B89b94389b9214cA476dd5107Af92a
Audited by	© cyberscope

Table of Contents

Review

Contract Name	MetaGold
Compiler Version	v0.8.17+commit.8df45f5f
Optimization	200 runs
Explorer	https://bscscan.com/address/0x1f77f06333b89b94389b9214ca476dd5107af92a
Address	0x1f77f06333b89b94389b9214ca476dd5107af92a
Network	BSC
Symbol	MGF
Decimals	18
Total Supply	21,000,000

Audit Updates

Initial Audit	31 Jan 2023

Source Files

Filename	SHA256
MetaGold.sol	577f07dcdb221dfd76e27b36451b3eed1442b46ef1215061d1445140665c02c8

Roles

The contract consists of four roles.

`Owner` is responsible for configuring the smart contract.

`MetaAdmin` is responsible for removing fees, recovering tokens, and configuring Lockpay contract.

`StakingContract` is responsible for minting and burning tokens.

`LockpayRebase` is responsible for manual LockpayRebase.

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

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

BT - Burns Tokens

Criticality	Critical
Status	Unresolved

Description

The contract owner has the authority to burn tokens from a specific address. The stake contract may take advantage of it by calling the `burn` function. As a result, the targeted address will lose the corresponding tokens.

```
function burnSTAKE(uint256 amount) public onlyStakingContract {
  //prevent stakingContract address from being normal wallet
  require(stakingContract.testStakingContract(), "burnSTAKE: staking
contract address is not valid");
  super._burn(stakingContractAddress,amount);
}
```

Recommendation

The team should carefully manage the staking address. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract burn functions.

MT - Mints Tokens

Criticality	Critical
Location	MetaGold.sol#L1171
Status	Unresolved

Description

The `onlyStakingContract` has the authority to mint tokens. The staking address may take advantage of it by calling the `mintSTAKE` function. As a result, the contract tokens will be highly inflated.

```
function mintSTAKE(uint256 amount) public onlyStakingContract {
  //prevent stakingContract address from being normal wallet
  require(stakingContract.testStakingContract(), "mintSTAKE: staking
contract address is not valid");
  super._mint(stakingContractAddress,amount);
}
```

Recommendation

The team should carefully manage the staking address. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

ELFM - Exceeds Fees Limit

Criticality	Critical
Location	MetaGold.sol#L990,995
Status	Unresolved

Description

The contract owner has the authority to increase over the allowed limit of 25%. The owner may take advantage of it by calling the `setBurnFee` and `setMarketingFee` functions with a high percentage value.

```
function setBurnFee(uint256 value) external onlyOwner {
  require(value >= 0, "Fee must be greater than zero");
  burnFee = value;
}

function setMarketingFee(uint256 value) external onlyOwner {
  require(value >= 0, "Fee must be greater than zero");
  marketingFee = value;
}
```

Recommendation

The contract could embody a check for the maximum acceptable value. 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. That risk can be prevented by temporarily locking the contract or renouncing ownership.

ULTW - Transfers Liquidity to Team Wallet

Criticality	Minor / Informative
Location	MetaGold.sol#L1258,1270,1274
Status	Unresolved

Description

The contract owner has the authority to transfer funds without limit to the team wallet. These funds have been accumulated from fees collected from the contract. The owner may take advantage of it by calling the `swapAndSendToFee`, `sendBNBtoWallet`, and `sendAllBNBToWallet` methods.

```
function swapAndSendToFee(uint256 tokens) private  {
    uint256 initialBalance = getContractBNBBalance();
    _swapTokensForBNB(tokens);
    transferOutBNB(payable(directPaymentBNBAddressReceiver),
getContractBNBBalance().sub(initialBalance));
}

function sendBNBtoWallet(uint256 bnbBalance) public onlyOwner {
    transferOutBNB(payable(directPaymentBNBAddressReceiver),
bnbBalance);
}

function sendAllBNBToWallet() public onlyOwner {
    transferOutBNB(payable(directPaymentBNBAddressReceiver),
getContractBNBBalance());
}
```

Recommendation

The contract could embody a check for the maximum amount of funds that can be swapped. Since a huge amount may volatile the token's price. 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. That risk can be prevented by temporarily locking the contract or renouncing ownership.

ST - Stops Transactions

Criticality	Critical
Location	MetaGold.sol#L1193,1210
Status	Unresolved

Description

The contract owner has the authority to stop the sales for all users excluding the owner. The owner may take advantage of it by utilizing the `protector` or contract. As a result, the contract may operate as a honeypot.

```
if( protectorEnabled) {
  require(protector.processSell(from,to,amount), "Protector: Failed to
sell");
}
```

The contract owner has the authority to stop the sales for all users. The owner may take advantage of it by utilizing the `LockpayContract` contract.

```
if(nextLockpayRebase > 0 && nextLockpayRebase < block.timestamp) {
  LockpayContract.manualRebase();
  nextLockpayRebase = nextLockpayRebase.add(lockpayTwentyFourHours);
  lastLockpayRebase = block.timestamp;
}
```

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. That risk can be prevented by temporarily locking the contract or renouncing ownership.

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	US	Untrusted Source	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	BLC	Business Logic Concern	Unresolved
●	CR	Code Repetition	Unresolved
●	MVN	Misleading Variables Naming	Unresolved
●	CO	Code Optimization	Unresolved
●	TUU	Time Units Usage	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L06	Missing Events Access Control	Unresolved
●	L07	Missing Events Arithmetic	Unresolved

●	L08	Tautology or Contradiction	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

US - Untrusted Source

Criticality	Critical
Location	MetaGold.sol#L683,735,736
Status	Unresolved

Description

The contract uses an external contract in order to determine the transaction's flow. The external contract is untrusted. As a result, it may produce security issues and harm the transactions.

```
IProtector private protector
address private LockpayContractAddress;
ILockpayContract private LockpayContract;
```

Recommendation

The contract should use a trusted external source. A trusted source could be either a commonly recognized or an audited contract. The pointing addresses should not be able to change after the initialization.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	MetaGold.sol#L1227
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `swapTokensAtAmount` sets 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. Additionally, the variable `marketingSwapMultiplier` could produce the same issue.

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( feesEnabled) {
    uint256 contractTokenBalance = getContractTokenBalance();
    bool canSwap = contractTokenBalance > swapTokensAtAmount;

    if( canSwap &&
        !swapping &&
        !automatedMarketMakerPairs[from]
    ) {
        swapping = true;
        if(marketingSwapMultiplier > 0) {
            uint256 marketingTokens =
calcMarketingTokens(contractTokenBalance);
            swapAndSendToFee(marketingTokens);
        }
        swapping = false;
    }
```

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.

BLC - Business Logic Concern

Criticality	Minor / Informative
Location	MetaGold.sol#L885
Status	Unresolved

Description

The implementation may not follow the expected behavior. The contract modifies the `directPaymentBNBAddressReceiver` address and it exludes the address from fees. Without including the previous `directPaymentBNBAddressReceiver` to the fees.

```
function updateDirectPaymentBNBAddressReceiver(address newAddress)
public onlyOwner {
    directPaymentBNBAddressReceiver = newAddress;
    _isExcludedFromFees[newAddress] = true;
}
```

Recommendation

The team is advised to carefully check if the implementation follows the expected business logic. It is recommended to include previous address to the fees if there are exempted.D

CR - Code Repetition

Criticality	Minor / Informative
Location	MetaGold.sol#L923,932,955,972,1283,1296
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 recoverTokens(address tokenAddress, address receiver) external
onlyOwner {
    IERC20(tokenAddress).approve(address(this), MAX_INT);
    IERC20(tokenAddress).transferFrom(
                        address(this),
                        receiver,
                        IERC20(tokenAddress).balanceOf(address(this))
    );
}

function recoverTokensMetaAdmin(address tokenAddress, address receiver)
external onlyMetaAdmin {
    IERC20(tokenAddress).approve(address(this), MAX_INT);
    IERC20(tokenAddress).transferFrom(
                        address(this),
                        receiver,
                        IERC20(tokenAddress).balanceOf(address(this))
    );
}

function setLockpayContractInfo(address lpaddress, uint256 lockpay24,
bool enabled) public onlyOwner {
    LockpayContract = ILockpayContract(lpaddress);
    LockpayContractAddress = lpaddress;
    lockpayTwentyFourHours = lockpay24;
    doLockpayRebase = enabled;
}

function setLockpayContractInfoMeta(address lpaddress, uint256
lockpay24, bool enabled) public onlyMetaAdmin {
    LockpayContract = ILockpayContract(lpaddress);
    LockpayContractAddress = lpaddress;
    lockpayTwentyFourHours = lockpay24;
    doLockpayRebase = enabled;
}

function _swapTokensForBNB(uint256 tokenAmount) private {
    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = uniswapV2Router.WETH();

    uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
        tokenAmount,
        0,
        path,
        address(this),
        block.timestamp
    );
}
```

```
function swapTokensForBNB(uint256 tokenAmount) public onlyOwner {
    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = uniswapV2Router.WETH();

    uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
        tokenAmount,
        0,
        path,
        address(this),
        block.timestamp
    );
}
```

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.

MVN - Misleading Variables Naming

Criticality	Minor / Informative
Location	MetaGold.sol#L735,958,975
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 variable lockpayTwentyFourHours illustrate 24 hours but it can be mutated.

```
uint256 private lockpayTwentyFourHours = 86400;

lockpayTwentyFourHours = lockpay24;
```

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.

CO - Code Optimization

Criticality	Minor / Informative
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The state variable `meta_address` is unnecessary. This information can be obtained through the built-in `address(this)` expression, which returns the address of the current contract

```
meta_address = address(this);

IERC20(meta_address).transferFrom(
    tokenWalletForSendingMETAfrom,
    msg.sender,
    paidMetaGold
);

IERC20(tokenAddress).approve(address(this), MAX_INT);
IERC20(tokenAddress).transferFrom(
    address(this),
    receiver,
    IERC20(tokenAddress).balanceOf(address(this))
);

...
```

The smart contract maintains multiple copies of the same information through the use of duplicate variables. For instance, `stakingContract` and `stakingContractAddress`

```
stakingContract
stakingContractAddress
LockpayContract = ILockpayContract(lpaddress);
LockpayContractAddress = lpaddress;
```

The contract is using the interface reference to call the smart contract. As a result, it increases gas costs and longer transaction times, as it requires additional computational steps to resolve the correct function call.

```
IERC20(meta_address).transferFrom(
    tokenWalletForSendingMETAfrom,
    msg.sender,
    reward
);
```

Recommendation

The team is advised to take into consideration these segments and rewrite them so the runtime will be more performant.

- It's recommended to use built-in expressions instead of a separate state variable.
- It is advisable to adopt the use of a single authoritative source of information within the smart contract.
- It is recommended to directly call the transferFrom function without using the interface reference.

That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

TUU - Time Units Usage

Criticality	Minor / Informative
Location	MetaGold.sol#L735
Status	Unresolved

Description

The contract is using arbitrary numbers to form time-related values. As a result, it decreases the readability of the codebase and prevents the compiler to optimize the source code.

```
uint256 private lockpayTwentyFourHours = 86400;
```

Recommendation

It is a good practice to use the time units reserved keywords like `seconds`, `minutes`, `hours`, `days`, `weeks` and `years` to process time-related calculations.

It's important to note that these time units are simply a shorthand notation for representing time in seconds, and do not have any effect on the actual passage of time or the execution of the contract. The time units are simply a convenience for expressing time in a more human-readable form.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	MetaGold.sol#L92,138
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 SafeMathInt {...}
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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	MetaGold.sol#L702
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 MAX_INT =
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
```

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	MetaGold.sol#L666,669,702,731,732,1070,1098,1101,1105,1317,1509,1510,1527
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 bnb_address
address public meta_address
uint256 private MAX_INT =
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
address private LockpayContractAddress
ILockpayContract private LockpayContract
uint256 _feeMultiplier
bool _feesEnabled
bool _block
bool _send
function WETH() external pure returns (address);
function DOMAIN_SEPARATOR() external view returns (bytes32);
function PERMIT_TYPEHASH() external pure returns (bytes32);
function MINIMUM_LIQUIDITY() external pure returns (uint);
```

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	MetaGold.sol#L94
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.

```
int256 private constant MAX_INT256 = ~(int256(1) << 255)
```

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.

L06 - Missing Events Access Control

Criticality	Minor / Informative
Location	MetaGold.sol#L950,1118
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. There are functions that have no event emitted, so it is difficult to track off-chain changes.

```
stakingContractAddress = sc
lockpayRebaseAddressAdmin = newAddress
```

Recommendation

To avoid this issue, 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.

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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	MetaGold.sol#L958,969,992,997,1002,1006,1010,1067,1071
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.

```
lockpayTwentyFourHours = lockpay24
nextLockpayRebase = nr
burnFee = value
marketingFee = value
bonusDirectTransaction = value
bonusSwapTransactionToReferrer = value
bonusSwapTransactionToReferral = value
swapTokensAtAmount = amount
marketingSwapMultiplier = _feeMultiplier
```

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	MetaGold.sol#L991,996,1001,1005,1009
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.

```
require(value >= 0, "Fee must be greater than zero")
require(value >= 0, "Bonus direct transaction must be greater than
zero")
```

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	MetaGold.sol#L124,130,193
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 abs(int256 a) internal pure returns (int256) {
        require(a != MIN_INT256);
        return a < 0 ? -a : a;
    }

function toUint256Safe(int256 a) internal pure returns (uint256) {
...
    }

function toInt256Safe(uint256 a) internal pure returns (int256) {
    int256 b = int256(a);
    require(b >= 0);
    return b;
  }
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	MetaGold.sol#L817,845,950,957,974,1055,1063,1118
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.

```
wallet1.transfer(msg.value)
stakingContractAddress = sc
LockpayContractAddress = lpaddress
directPaymentBNBAddressReceiver = newAddress
tokenWalletForSendingMETAfrom = newAddress
lockpayRebaseAddressAdmin = newAddress
```

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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	MetaGold.sol#L809,837,870,884,925,934
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(meta_address).transferFrom(
                            tokenWalletForSendingMETAfrom,
                            msg.sender,
                            paidMetaGold
                        )

IERC20(meta_address).transferFrom(
                        tokenWalletForSendingMETAfrom,
                        msg.sender,
                        paidMetaGold
                    )

...
```

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**
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Meta	Interface	IERC20		
	tokenPair	External		-
SafeMathInt	Library			
	mul	Internal		
	div	Internal		
	sub	Internal		
	add	Internal		
	abs	Internal		
	toUint256Safe	Internal		
SafeMath	Library			

	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
	mod	Internal		
	mod	Internal		
SafeMathUint	Library			
	toInt256Safe	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
MetaAdmin	Implementation	Context		
		Public	✓	-
	metaAdmin	Public		-
ILockpayContract	Interface			
	manualRebase	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-

ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_spendAllowance	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
ReentrancyGuard	Implementation			
		Public	✓	-
MetaGold	Implementation	ERC20, Ownable, ReentrancyGuard, MetaAdmin		

	getLockpayInfo	Public		-
		Public	✓	ERC20
		External	Payable	-
	buyFromOwner	Public	Payable	-
	buySwap	Public	Payable	-
	calculateRewards	Public		-
	recoverTokens	External	✓	onlyOwner
	recoverTokensMetaAdmin	External	✓	onlyMetaAdmin
	excludeFromFees	Public	✓	onlyOwner
	setStakingContract	Public	✓	onlyOwner
	setLockpayContractInfo	Public	✓	onlyOwner
	getBlockTimestamp	Public		-
	setNextLockpayRebase	Public	✓	onlyOwner
	setLockpayContractInfoMeta	Public	✓	onlyMetaAdmin
	getPriceBNB	Public		-
	setBurnFee	External	✓	onlyOwner
	setMarketingFee	External	✓	onlyOwner
	setBonusDirectTransaction	External	✓	onlyOwner
	setBonusSwapTransactionReferrer	External	✓	onlyOwner
	setBonusSwapTransactionReferral	External	✓	onlyOwner
	setBonusReferralSwap	Public	✓	onlyOwner
	setBonusReferrerSwap	Public	✓	onlyOwner
	getBonusReferralSwap	Public		-
	getBonusReferrerSwap	Public		-
	setAutomatedMarketMakerPair	Public	✓	onlyOwner
	_setAutomatedMarketMakerPair	Private	✓	
	isMustDoFee	Public		-
	manualLockpayRebase	Public	✓	onlyLockpayRebase

	updateDirectPaymentBNBAddressReceiver	Public	✓	onlyOwner
	isExcludedFromFees	Public		-
	updateTokenWalletForSendingMETAfrom	Public	✓	onlyOwner
	setSwapTokensAtAmount	External	✓	onlyOwner
	setMarketingSwapMultiplier	External	✓	onlyOwner
	setMustDoFee	Public	✓	onlyOwner
	calcMarketingTokens	Private		
	getContractTokenBalance	Public		-
	swapOnDemand	External	✓	onlyOwner
	setFeesEnabled	Public	✓	onlyOwner
	setBlockAllIncoming	Public	✓	onlyOwner
	setSendMetaGoldIncoming	Public	✓	onlyOwner
	setProtector	Public	✓	onlyOwner
	setProtectorEnabled	Public	✓	onlyOwner
	setLockpayRebaseAddressAdmin	Public	✓	onlyOwner
	removeAllFees	Public	✓	onlyMetaAdmin
	showMyLimit	Public		-
	getTradeData	Public		-
	getTradeData2	Public		-
	whoAmIProtected	Public		-
	whoAmIProtected2	Public		-
	getProtectorVersion	Public		-
	getProtector24hrs	Public		-
	destroyTokensFromOwnerWallet	Public	✓	onlyOwner
	burnSTAKE	Public	✓	onlyStakingContract
	mintSTAKE	Public	✓	onlyStakingContract
	_transfer	Internal	✓	
	getMetaGoldVersion	Public		-

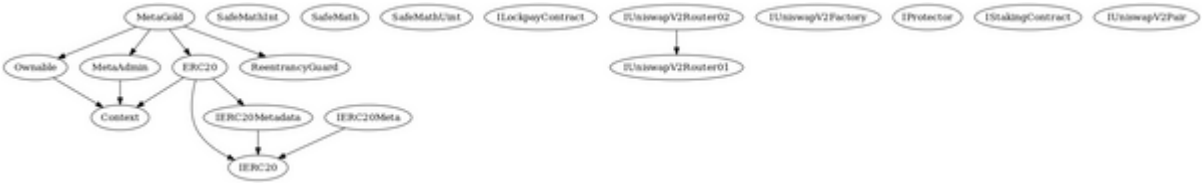
	swapAndSendToFee	Private	✓	
	getContractBNBBalance	Public		-
	sendBNBtoWallet	Public	✓	onlyOwner
	sendAllBNBToWallet	Public	✓	onlyOwner
	transferOutBNB	Private	✓	
	_swapTokensForBNB	Private	✓	
	swapTokensForBNB	Public	✓	onlyOwner
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	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	✓	-
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-
IProtector	Interface			
	getMaxTokenAllowedToSell	External		-
	getMyAddress	External		-
	getMyAddressProtected	External		-
	getVersion	External		-
	getTwentyFourHours	External		-
	processSell	External	✓	-
	getTradeData	External		-
	getTradeData2	External		-

IStakingContract	Interface			
	testStakingContract	External		-
	addReward	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		-
	mint	External	✓	-
	burn	External	✓	-
	swap	External	✓	-

	skim	External	✓	-
	sync	External	✓	-
	initialize	External	✓	-

Inheritance Graph



Flow Graph

Summary

There are some functions that can be abused by the owner like stopping transactions, manipulating the fees, transferring funds to the team's wallet, mint tokens, and burning tokens from any address. The contract can be converted into a honeypot and prevent users from selling if the owner abuses the admin functions. if the contract owner abuses the mint functionality, the contract will be highly inflated. if the contract owner abuses the burning functionality, then the users could lose their tokens. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats.

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