Cyberscope

Audit Report

BetBitcoin.ai

October 2023

Network	BSC
Address	0x8eed289c760a521615644fb51519ecb8056c0c9e
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	Unresolved
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	RRS	Redundant Require Statement	Unresolved
●	RC	Redundant Calculations	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	DDP	Decimal Division Precision	Unresolved
●	OCTD	Transfers Contract's Tokens	Unresolved
●	AOI	Arithmetic Operations Inconsistency	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved

●	L15	Local Scope Variable Shadowing	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Review

Contract Name	BetBitcoinAi
Compiler Version	v0.8.9+commit.e5eed63a
Optimization	200 runs
Explorer	https://bscscan.com/address/0x8eed289c760a521615644fb51519ecb8056c0c9e
Address	0x8eed289c760a521615644fb51519ecb8056c0c9e
Network	BSC
Symbol	BB
Decimals	18
Total Supply	400,000,000

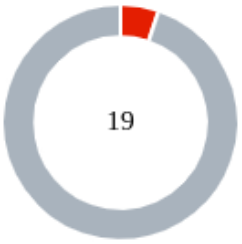
Audit Updates

Initial Audit	13 Oct 2023

Source Files

Filename	SHA256
BetBitcoinAi.sol	286cc64513e632561579ba6e3165dee4305f2bf4acf7075d24b72a41b638d41b

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
Location	BetBitcoinAi.sol#L610,703,749
Status	Unresolved

Description

The transactions are initially invoked for all users excluding the authorized addresses. The owner can enable the transactions for all users. Once the transactions are enable the owner will not be able to disable them again.

```
if (!isTrading) {
  require(_isExcludedFromFees[sender] ||
_isExcludedFromFees[recipient], "Trading is not active.");
}
```

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. Some suggestions are:

- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

ELFM - Exceeds Fees Limit

Criticality	Minor / Informative
Location	BetBitcoinAi.sol#L610,749
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 `swapTrading` function with a high percentage value.

Specifically, the contract owner has the authority to repetitively invoke the `swapTrading` function, resulting to a high fees value of `99%` for the next 2 blocks. As a result, users who interact with the contract within these 2 blocks from when `swapTrading` is triggerd will be charged a total of `99%` fees.

```
function swapTrading() external onlyOwner {
        isTrading = true;
        swapEnabled = true;
        taxTill = block.number + 2;
    }
...

if(block.number < taxTill) {
    fees = amount.mul(99).div(100);
    tokensForMarketing += (fees * 94) / 99;
    tokensForDev += (fees * 5) / 99;
}
```

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.

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.

RRS - Redundant Require Statement

Criticality	Minor / Informative
Location	BetBitcoinAi.sol#L191
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.

RC - Redundant Calculations

Criticality	Minor / Informative
Location	BetBitcoinAi.sol#L577,626,634
Status	Unresolved

Description

The contract calculates values for `maxBuyAmount`, `maxSellAmount`, and `maxWalletAmount` by multiplying the `totalSupply` by `100` and then dividing by `100`. This essentially results in the same value as `totalSupply` without any change. Additionally, the `require` statements are performing calculations that could be simplified to be more efficient. As a result these calculations are redundant and gas inefficient.

```
maxBuyAmount = totalSupply * 100 / 100; // 100% maxTransactionAmountTxn
maxSellAmount = totalSupply * 100 / 100; // 100% maxTransactionAmountTxn
maxWalletAmount = totalSupply * 100 / 100; // 100% maxWallet
...
require(((totalSupply() * newMaxBuy) / 1000) >= (totalSupply() / 100),
"maxBuyAmount must be higher than 1%");
require(((totalSupply() * newMaxSell) / 1000) >= (totalSupply() / 100),
"maxSellAmount must be higher than 1%");
...
require(((totalSupply() * newPercentage) / 1000) >= (totalSupply() /
100), "Cannot set maxWallet lower than 1%");
```

Recommendation

It is recommended to simplify the calculations by directly assigning the `totalSupply` to `maxBuyAmount`, `maxSellAmount`, and `maxWalletAmount` without the redundant multiplication and division. Furthermore, the conditions in the require statements can be optimized to avoid unnecessary calculations and improve gas efficiency.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	BetBitcoinAi.sol#L619
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `thresholdSwapAmount` 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.

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 updateThresholdSwapAmount(uint256 newAmount) external
onlyOwner returns(bool){
        thresholdSwapAmount = newAmount;
        return true;
    }
```

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.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	BetBitcoinAi.sol#L750,754,761
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.

```
    fees = amount.mul(99).div(100);
     tokensForMarketing += (fees * 94) / 99;
     tokensForDev += (fees * 5) / 99;
} else if (marketPair[recipient] && _fees.sellTotalFees > 0) {
     fees = amount.mul(_fees.sellTotalFees).div(100);
     tokensForLiquidity += fees * _fees.sellLiquidityFee /
_fees.sellTotalFees;
     tokensForMarketing += fees * _fees.sellMarketingFee /
_fees.sellTotalFees;
     tokensForDev += fees * _fees.sellDevFee /
_fees.sellTotalFees;
     }
// on buy
else if (marketPair[sender] && _fees.buyTotalFees > 0) {
     fees = amount.mul(_fees.buyTotalFees).div(100);
     tokensForLiquidity += fees * _fees.buyLiquidityFee /
_fees.buyTotalFees;
     tokensForMarketing += fees * _fees.buyMarketingFee /
_fees.buyTotalFees;
     tokensForDev += fees * _fees.buyDevFee /
_fees.buyTotalFees;
     }
```

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.

OCTD - Transfers Contract's Tokens

Criticality	Minor / Informative
Location	BetBitcoinAi.sol#L672
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 `rescueERC20` function.

```
    function rescueERC20(address tokenAdd, uint256 amount)
external onlyOwner {
        IERC20(tokenAdd).transfer(owner(), amount);
    }
```

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.

AOI - Arithmetic Operations Inconsistency

Criticality	Minor / Informative
Location	BetBitcoinAi.sol#L807,819,827
Status	Unresolved

Description

The contract uses both the SafeMath library and native arithmetic operations. The SafeMath library is commonly used to mitigate vulnerabilities related to integer overflow and underflow issues. However, it was observed that the contract also employs native arithmetic operators (such as +, -, *, /) in certain sections of the code.

The combination of SafeMath library and native arithmetic operations can introduce inconsistencies and undermine the intended safety measures. This discrepancy creates an inconsistency in the contract's arithmetic operations, increasing the risk of unintended consequences such as inconsistency in error handling, or unexpected behavior.

```
uint256 toSwap = tokensForLiquidity + tokensForMarketing +
tokensForDev;
...
uint256 amountToSwapForETH =
contractTokenBalance.sub(liquidityTokens);
...
uint256 ethForMarketing =
newBalance.mul(tokensForMarketing).div(toSwap);
uint256 ethForDev = newBalance.mul(tokensForDev).div(toSwap);
uint256 ethForLiquidity = newBalance - (ethForMarketing +
ethForDev);
```

Recommendation

To address this finding and ensure consistency in arithmetic operations, it is recommended to standardize the usage of arithmetic operations throughout the contract. The contract should be modified to either exclusively use SafeMath library functions or entirely rely on native arithmetic operations, depending on the specific requirements and design

considerations. This consistency will help maintain the contract's integrity and mitigate potential vulnerabilities arising from inconsistent arithmetic operations.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	BetBitcoinAi.sol#L653,676
Status	Unresolved

Description

The contract modifies the state of the following variables without checking if their current value is the same as the one given as an argument. As a result, the contract performs redundant storage writes, when the provided parameter matches the current state of the variables, leading to unnecessary gas consumption and inefficiencies in contract execution.

```
   function excludeFromFees(address account, bool excluded)
public onlyOwner {
        _isExcludedFromFees[account] = excluded;
    }
    function excludeFromWalletLimit(address account, bool
excluded) public onlyOwner {
        _isExcludedMaxWalletAmount[account] = excluded;
    }
    function excludeFromMaxTransaction(address updAds, bool
isEx) public onlyOwner {
        _isExcludedMaxTransactionAmount[updAds] = isEx;
    }
    ...
   function setWallets(address _marketingWallet,address
_devWallet) external onlyOwner{
        marketingWallet = _marketingWallet;
        devWallet = _devWallet;
    }
```

Recommendation

The team is advised to implement additional checks within to prevent redundant storage writes when the provided argument matches the current state of the variables. By incorporating statements to compare the new values with the existing values before

proceeding with any state modification, the contract can avoid unnecessary storage operations, thereby optimizing gas usage.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	BetBitcoinAi.sol#L638
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
   function updateFees(uint8 _marketingFeeBuy, uint8
_liquidityFeeBuy,uint8 _devFeeBuy,uint8 _marketingFeeSell,
uint8 _liquidityFeeSell,uint8 _devFeeSell) external onlyOwner{
        _fees.buyMarketingFee = _marketingFeeBuy;
        _fees.buyLiquidityFee = _liquidityFeeBuy;
        _fees.buyDevFee = _devFeeBuy;
        _fees.buyTotalFees = _fees.buyMarketingFee +
_fees.buyLiquidityFee + _fees.buyDevFee;

        _fees.sellMarketingFee = _marketingFeeSell;
        _fees.sellLiquidityFee = _liquidityFeeSell;
        _fees.sellDevFee = _devFeeSell;
        _fees.sellTotalFees = _fees.sellMarketingFee +
_fees.sellLiquidityFee + _fees.sellDevFee;
        require(_fees.buyTotalFees <= 10, "Must keep fees at
10% or less");
        require(_fees.sellTotalFees <= 10, "Must keep fees at
10% or less");

    }
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such

as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	BetBitcoinAi.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.

```
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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	BetBitcoinAi.sol#L346,515,535,536,634,672
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);

Fees public _fees = Fees({
        buyTotalFees: 0,
        buyMarketingFee: 0,
        buyDevFee:0,
        buyLiquidityFee: 0,

        sellTotalFees: 0,
        sellMarketingFee: 0,
        sellDevFee:0,
        sellLiquidityFee: 0
    })
mapping(address => bool) public _isExcludedMaxTransactionAmount
mapping(address => bool) public _isExcludedMaxWalletAmount

...
```

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	BetBitcoinAi.sol#L281
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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	BetBitcoinAi.sol#L616,624,631
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.

```
thresholdSwapAmount = newAmount
maxBuyAmount = (totalSupply() * newMaxBuy) / 1000
maxWalletAmount = (totalSupply() * newPercentage) / 1000
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	BetBitcoinAi.sol#L323,329,336
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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	BetBitcoinAi.sol#L746,747,748,750,751,752,753,757,758,759,760
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.

```
fees = amount.mul(_fees.buyTotalFees).div(100)
tokensForDev += fees * _fees.buyDevFee / _fees.buyTotalFees
```

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	BetBitcoinAi.sol#L571
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.

```
uint256 totalSupply = 400000000 * 1e18
```

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	BetBitcoinAi.sol#L673,674
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.

```
marketingWallet = _marketingWallet
devWallet = _devWallet
```

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	BetBitcoinAi.sol#L669
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(tokenAdd).transfer(owner(), 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**
IUniswapV2Factory	Interface			
	createPair	External	✓	-
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
Context	Implementation			
	_msgSender	Internal		

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	✓	
	_approve	Internal	✓	
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		

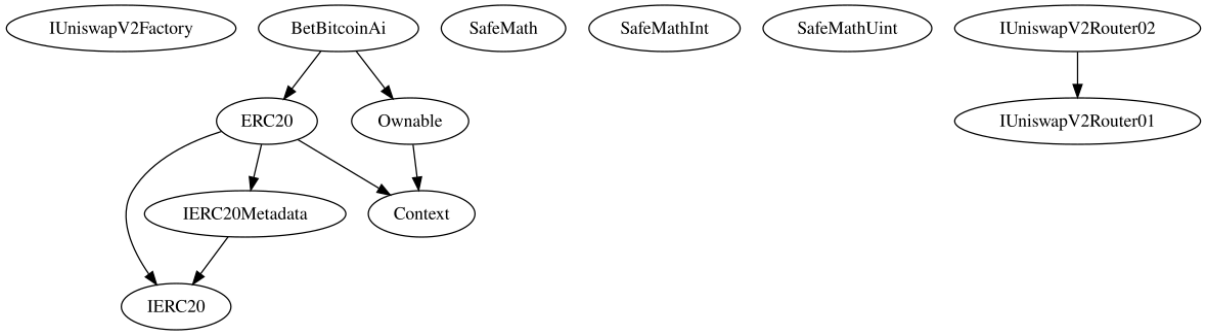
	mul	Internal		
	div	Internal		
	div	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
SafeMathInt	Library			
	mul	Internal		
	div	Internal		
	sub	Internal		
	add	Internal		
	abs	Internal		
	toUint256Safe	Internal		
SafeMathUint	Library			
	toInt256Safe	Internal		
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	IUniswapV2 Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-

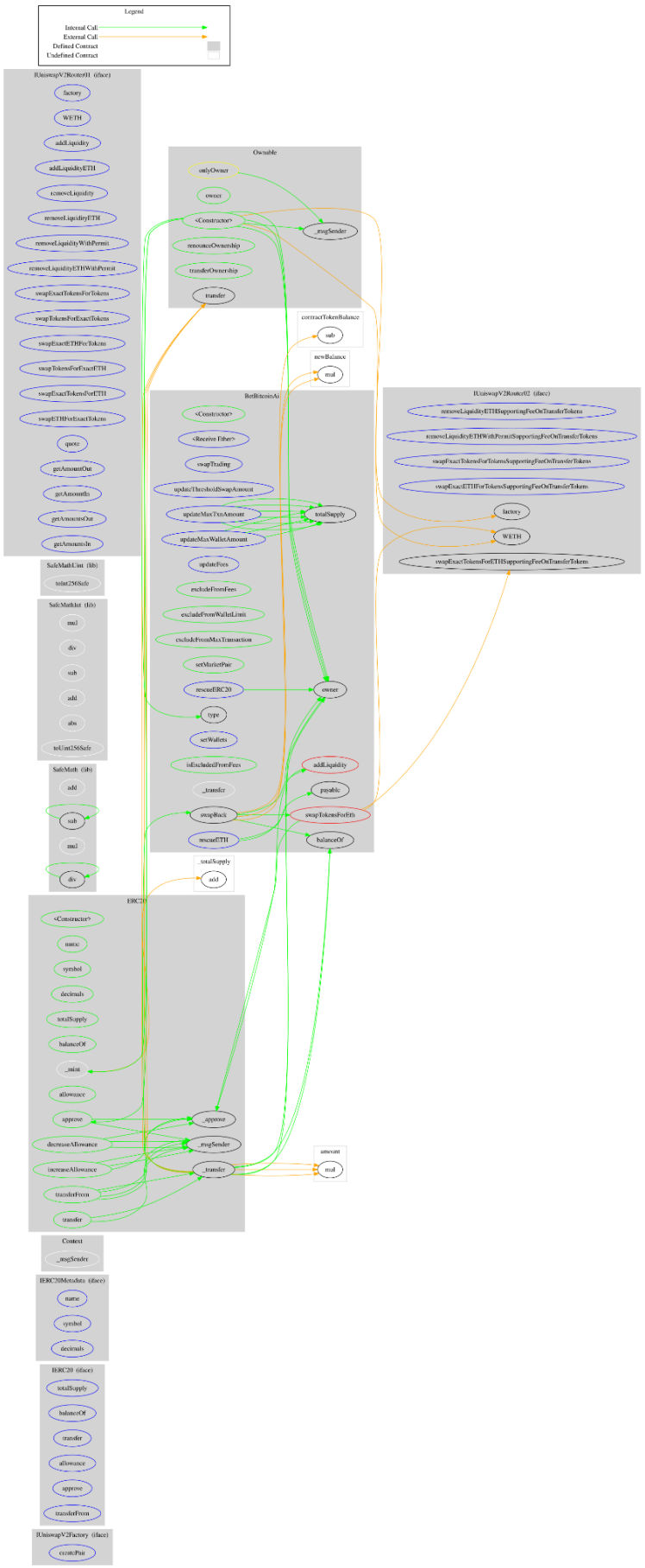
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
BetBitcoinAi	Implementation	ERC20, Ownable		
		Public	✓	ERC20
		External	Payable	-
	swapTrading	External	✓	onlyOwner
	updateThresholdSwapAmount	External	✓	onlyOwner
	updateMaxTxnAmount	External	✓	onlyOwner
	updateMaxWalletAmount	External	✓	onlyOwner
	updateFees	External	✓	onlyOwner
	excludeFromFees	Public	✓	onlyOwner
	excludeFromWalletLimit	Public	✓	onlyOwner
	excludeFromMaxTransaction	Public	✓	onlyOwner
	setMarketPair	Public	✓	onlyOwner
	rescueETH	External	✓	onlyOwner
	rescueERC20	External	✓	onlyOwner
	setWallets	External	✓	onlyOwner
	isExcludedFromFees	Public		-
	_transfer	Internal	✓	
	swapTokensForEth	Private	✓	

	addLiquidity	Private	✓	
	swapBack	Private	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IUniswapV2Router01 (iface)
factory
WETH
addLiquidity
addLiquidityETH
removeLiquidity
removeLiquidityETH
removeLiquidityWithPermit
removeLiquidityETHWithPermit
swapExactTokensForTokens
swapTokensForExactTokens
swapExactETHForTokens
swapTokensForExactETH
swapExactTokensForETH
swapETHForExactTokens
quote
getAmountOut
getAmountIn
getAmountsOut
getAmountsIn
Ownable
onlyOwner
owner
<Constructor>
_msgSender
renounceOwnership
transferOwnership
transfer
contractTokenBalance
sub
newBalance
mul
BetBitcoinAi
<Constructor>
<Receive Ether>
swapTrading
updateThresholdSwapAmount
updateMaxTxnAmount
totalSupply
updateMaxWalletAmount
updateFees
excludeFromFees
excludeFromWalletLimit
excludeFromMaxTransaction
setMarketPair
rescueERC20
owner
type
setWallets
isExcludedFromFees
addLiquidity
_transfer
payable
swapBack
swapTokensForEth
rescueETH
balanceOf
_totalSupply
add
IUniswapV2Router02 (iface)
removeLiquidityETHSupportingFeeOnTransferTokens
removeLiquidityETHWithPermitSupportingFeeOnTransferTokens
swapExactTokensForTokensSupportingFeeOnTransferTokens
swapExactETHForTokensSupportingFeeOnTransferTokens
factory
WETH
swapExactTokensForETHSupportingFeeOnTransferTokens
SafeMathUint (lib)
toInt256Safe
SafeMathInt (lib)
mul
div
sub
add
abs
toUint256Safe
SafeMath (lib)
add
sub
mul
div
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
_mint
allowance
approve
_approve
decreaseAllowance
_msgSender
increaseAllowance
_transfer
transferFrom
transfer
amount
mul
Context
_msgSender
IERC20Metadata (iface)
name
symbol
decimals
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
IUniswapV2Factory (iface)
createPair

Summary

BetBitcoin.ai 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