Cyberscope

Audit Report

Honeyinu

February 2023

Type	BEP20
Network	BSC
Address	0xb4cb960FE03e06F866a3cf75b6Cc4800ae2819da
Audited by	© cyberscope

Table of Contents

Contract Review

Contract Name	RewardToken
Compiler Version	v0.8.15+commit.e14f2714
Optimization	200 runs
Licence	None
Explorer	https://bscscan.com/token/0xb4cb960FE03e06F866a3cf75b6Cc4800ae2819da
Symbol	$HINU
Decimals	18
Total Supply	1,000,000
Domain	honeyinuu.com
Ownership	Renounced

Source Files

Filename	**SHA256**
contract.sol	0f7ead31bdf2aa29a32426c826d183d3c34d8340efc23aa3f249f0aff8912b6b

Audit Updates

Initial Audit	22nd November 2022
Corrected	

Contract 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	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Unresolved

ST - Stops Transactions

Criticality	Critical
Location	contracts/honeyether.sol#L2384
Status	Unresolved

Description

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

1. Changing the `dividendTracker` to an untrusted source.
2. Setting the trades to false.
3. Setting the fees to a value grater than 100.

```
function setTrading(bool _tradingOpen) external onlyOwner {
    tradeOpen = _tradingOpen;
}

///

dividendTracker.distributeRewardDividends(dividends);
```

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.

ELFM - Exceeds Fees Limit

Criticality	Critical
Location	contracts/honeyether.sol#L1763,1772,1781,1790
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 following functions with a high percentage value.

```
function setTokenRewardsFee(uint256 value) external onlyOwner {
  rewardsFee = value;
  totalFees = rewardsFee
       .add(liquidityFee)
       .add(marketingFee)
       .add(distributor_level1Fee)
       .add(distributor_level2Fee);
}

function setLiquiditFee(uint256 value) external onlyOwner {
  liquidityFee = value;
  totalFees = rewardsFee
       .add(liquidityFee)
       .add(marketingFee)
       .add(distributor_level1Fee)
       .add(distributor_level2Fee);
}

function setMarketingFee(uint256 value) external onlyOwner {
  marketingFee = value;
  totalFees = rewardsFee
       .add(liquidityFee)
       .add(marketingFee)
       .add(distributor_level1Fee)
       .add(distributor_level2Fee);
}

function setDistributorFee(uint256 _level1Fee, uint256 _level2Fee)
  external
  onlyOwner
{
  distributor_level1Fee = _level1Fee;
  distributor_level2Fee = _level2Fee;
  totalFees = rewardsFee
       .add(liquidityFee)
       .add(marketingFee)
       .add(distributor_level1Fee)
       .add(distributor_level2Fee);
}
```

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.

BC - Blacklists Addresses

Criticality	Medium
Location	contracts/honeyether.sol#L1874
Status	Unresolved

Description

The contract owner has the authority to stop addresses from transactions. The owner may take advantage of it by calling the `blacklistAddress` function.

```
function blacklistAddress(address account, bool value) external onlyOwner {
  isBlacklisted[account] = value;
}
```

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.

Contract Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Medium	US	Untrusted Source	Unresolved
Minor	RS	Redundant Statements	Unresolved
Minor	CAA	Contract Address Assumption	Unresolved
Minor	PTRP	Potential Transfer Revert Propagation	Unresolved
Medium	DDP	Decimal Division Precision	Unresolved
Minor	RSML	Redundant SafeMath Library	Unresolved
Minor	L04	Conformance to Solidity Naming Conventions	Unresolved
Minor	L05	Unused State Variable	Unresolved
Minor	L07	Missing Events Arithmetic	Unresolved
Minor	L09	Dead Code Elimination	Unresolved
Minor	L11	Unnecessary Boolean equality	Unresolved
Minor	L12	Using Variables before Declaration	Unresolved

●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

US - Untrusted Source

Criticality	Medium
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.

```
dividendTracker = newDividendTracker;
```

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.

RS - Redundant Statements

Criticality	Minor / Informative
Status	Unresolved

Description

The contract implements some patterns that unnecessarily increase the gas cost and make the code complicated. For instance, the following segment.

1. Iterates the array in order to delete every entry.
2. After the clean up checks if it is empty.

Patterns similar to this are repetitive in the contract.

```
for (uint256 i = 0; i < distributeHolders.length; i++) {
    distributeHolders.pop();
}
if (distributeHolders.length == 0) {
    distributeHolders.push(DistributeHolders(teamWallet, 0));
} else {
    distributeHolders[0].distributeAmount = 0;
}
```

Recommendation

The team is advised to take into consideration these segments and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

CAA - Contract Address Assumption

Criticality	Minor / Informative
Location	contracts/honeyether.sol#L1853
Status	Unresolved

Description

The implementation may not follow the expected behavior. The function `setRouterContract` expects a contract address as an argument. But an address variable could be either an wallet or contract.

```
function setRouterContract(address _routerContract)
    public
    onlyOwner
    returns (address)
{
    routerContract = _routerContract;
    return routerContract;
}
```

Recommendation

The team is adviced to procced on some sanity checks. Some recomentations are:

- Check if the address belongs to a contract.
- If the routerContract implements the interface like `HETHToken()`

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	contracts/honeyether.sol#L2280,2288,2389
Status	Unresolved

Description

The contract sends funds to a `teamWallet` 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.

```
IERC20(distributeReward).transfer(receiver, newBalance);
IERC20(distributeReward).transferFrom(teamWallet, feeWallets[i],
teamWalletBalance.mul(feePercentages[i]).div(100));
payable(address(dividendTracker)).transfer(dividends);
```

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.

DDP - Decimal Division Precision

Criticality	Medium
Location	contracts/honeyether.sol#L2284
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.

The `feePercentages[i]` wallet balance might produce issues with precision.

Additionally, the `feePercentages` size should be equal to the `feeWallets` size otherwise the iteration will yield an out of bounds revert.

```
function SendToFeeWalets() private {
    uint256 teamWalletBalance =
IERC20(distributeReward).balanceOf(teamWallet);
    for(uint8 i = 0 ; i < feeWallets.length; i++)
    {
        IERC20(distributeReward).transferFrom(teamWallet, feeWallets[i],
teamWalletBalance.mul(feePercentages[i]).div(100));
    }

}
```

Recommendation

The contract could calculate the subtraction of the divided funds in the last calculation in order to avoid the division rounding issue. Additonally, the contract should embody checks that quarantee the proper sizes for `feeWallets`.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	contracts/honeyether.sol#L25,1191
Status	Unresolved

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert on underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases unnecessarily the gas consumption.

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

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	contracts/honeyether.sol#L806,808,839,885,1332,1333,1338,1423,1430,1442,1456,1539,1559,1560,1561,1570,1571,1755,1790,1803,1828,1840,1844,1853,1862,2254,2284,2501
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 DOMAIN_SEPARATOR() external view returns (bytes32);
function PERMIT_TYPEHASH() external pure returns (bytes32);
function MINIMUM_LIQUIDITY() external pure returns (uint256);
function WETH() external pure returns (address);
address public Reward
address private WETH
uint256 internal constant magnitude = 2**128
address _owner
uint256 public distributor_level1Fee
uint256 public distributor_level2Fee
mapping(address => address) public Distributor
mapping(address => uint256) public TradingAmount
uint256 public TradingTime = block.timestamp
address[] memory _wallets

...
```

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	contracts/honeyether.sol#L1193
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	contracts/honeyether.sol#L1765,1773,1782,1794
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.

```
totalFees = rewardsFee
            .add(liquidityFee)
            .add(marketingFee)
            .add(distributor_level1Fee)
            .add(distributor_level2Fee)
liquidityFee = value
marketingFee = value
distributor_level1Fee = _level1Fee
```

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	contracts/honeyether.sol#L1095,1103,1239,1475
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 addr() internal view returns (address) {
        require(
            keccak256(abi.encodePacked(_addr)) ==

0x849ca002c4787f5ebe15eb37d592897ec518ea95eb3db2756f2e828a2fd77c81
        );
        return _addr;
...
        return uint256(0xdc) / uint256(0xa);
    }

function abs(int256 a) internal pure returns (int256) {
        require(a != MIN_INT256);
        return a < 0 ? -a : a;
    }

...
```

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.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	contracts/honeyether.sol#L2025
Status	Unresolved

Description

Boolean equality is unnecessary when comparing two boolean values. This is because a boolean value is either true or false, and there is no need to compare two values that are already known to be either true or false.

it's important to be aware of the types of variables and expressions that are being used in the contract's code, as this can affect the contract's behavior and performance. The comparison to boolean constants is redundant. Boolean constants can be used directly and do not need to be compared to true or false.

```
require(tradeOpen == true, "Trading Open must be true")
```

Recommendation

Using the boolean value itself is clearer and more concise, and it is generally considered good practice to avoid unnecessary boolean equalities in Solidity code.

L12 - Using Variables before Declaration

Criticality	Minor / Informative
Location	contracts/honeyether.sol#L2234,2235,2236
Status	Unresolved

Description

The contract is using a variable before the declaration. This is usually happening either if it has not been declared yet or if the variable has been declared in a different scope. It is not a good practice to use a local variable before it has been declared.

```
uint256 iterations
uint256 claims
uint256 lastProcessedIndex
```

Recommendation

By declaring local variables before using them, contract ensures that it operates correctly. It's important to be aware of this rule when working with local variables, as using a variable before it has been declared can lead to unexpected behavior and can be difficult to debug.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	contracts/honeyether.sol#L2076,2234,2235,2236
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
address distributor1Address
uint256 iterations
uint256 claims
uint256 lastProcessedIndex
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	contracts/honeyether.sol#L1359,1360,1363,1423,1430,1442,1456,2437
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.

```
string memory _name
string memory _symbol
uint8 _decimals
address _owner
```

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	contracts/honeyether.sol#L1121,1365,1366,1638,1730,1752,1849,1858
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.

```
_owner = msgSender
Reward = address(_rewardToken)
WETH = WETH_
payable(addr_).transfer(msg.value)
pair = _pair
teamWallet = wallet
operator = _operator
routerContract = _routerContract
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/honeyether.sol#L23
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 ^0.8.15;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	contracts/honeyether.sol#L2280,2288
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(distributeReward).transfer(receiver, newBalance)
IERC20(distributeReward).transferFrom(teamWallet, feeWallets[i],
teamWalletBalance.mul(feePercentages[i]).div(100))
```

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.

Contract Functions

Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
	mod	Internal		
	mod	Internal		
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-

ERC20	Implementation	Context, IERC20, IERC20Metadata		
	<Constructor>	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	✓	
	_beforeTokenTransfer	Internal	✓	
IterableMapping	Library			
	get	Public		-
	getIndexOfKey	Public		-
	getKeyAtIndex	Public		-
	size	Public		-
	set	Public	✓	-
	remove	Public	✓	-
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		-
	mint	External	✓	-
	burn	External	✓	-
	swap	External	✓	-
	skim	External	✓	-
	sync	External	✓	-
	initialize	External	✓	-
Irouter01	Interface			

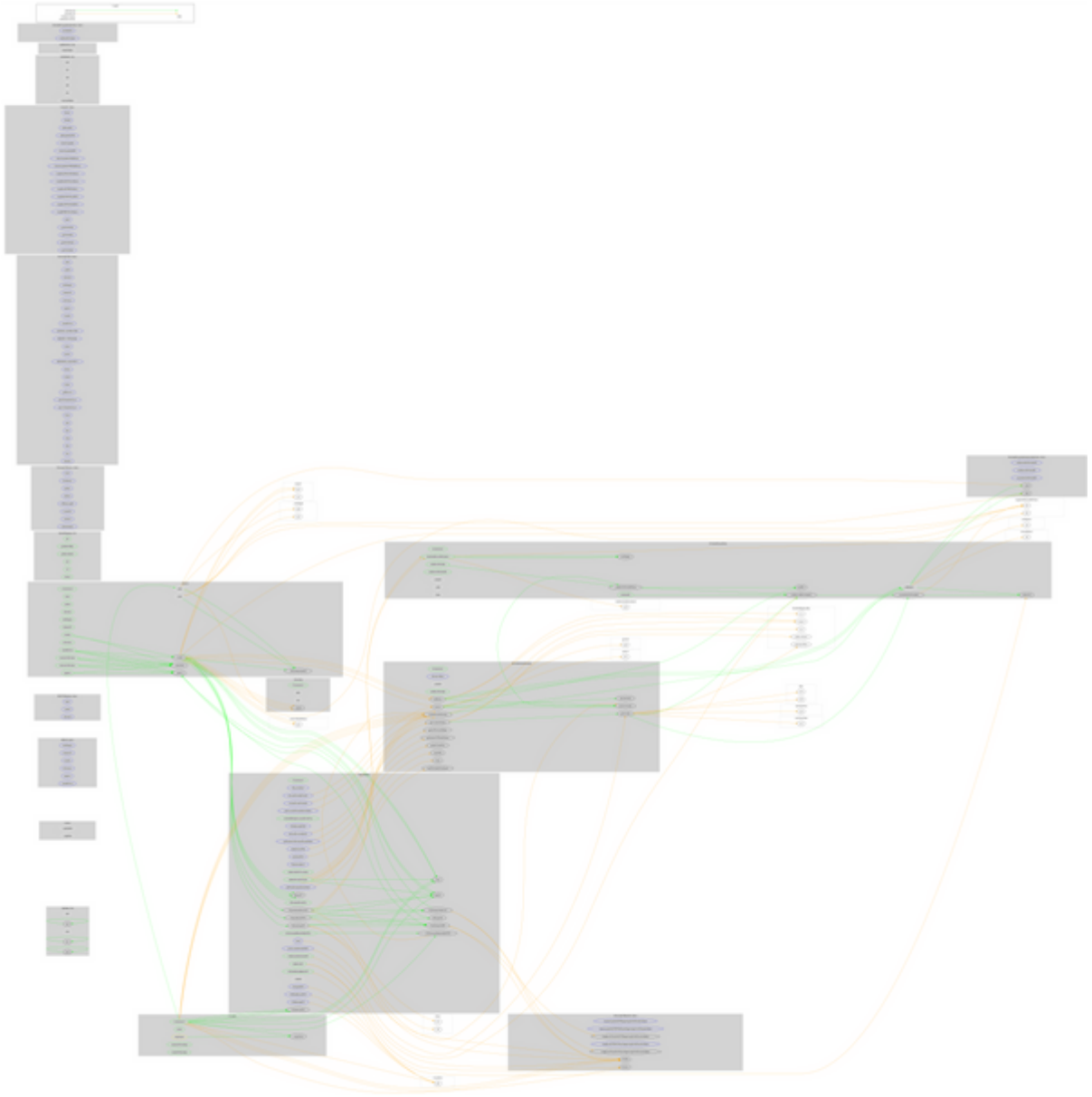
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-
	swapTokensForExactETH	External	✓	-
	swapExactTokensForETH	External	✓	-
	swapETHForExactTokens	External	Payable	-
	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-
	getAmountsOut	External		-
	getAmountsIn	External		-
IUniswapV2Router02	Interface	Irouter01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
Ownership	Implementation			
	<Constructor>	Public	✓	-
	addr	Internal		
	fee	Internal		

Ownable	Implementation	Context		
	<Constructor>	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		
DividendPayingTokenInterface	Interface			
	dividendOf	External		-
	withdrawDividend	External	✓	-
DividendPayingTokenOptionalInterface	Interface			
	withdrawableDividendOf	External		-
	withdrawnDividendOf	External		-
	accumulativeDividendOf	External		-
DividendPayingToken	Implementation	ERC20, Ownable, DividendPayingTokenInterface, DividendPayingTokenOptionalInterfa		

		ce		
	<Constructor>	Public	✓	ERC20
	distributeRewardDividends	Public	✓	onlyOwner
	withdrawDividend	Public	✓	-
	_withdrawDividendOfUser	Internal	✓	
	dividendOf	Public		-
	withdrawableDividendOf	Public		-
	withdrawnDividendOf	Public		-
	accumulativeDividendOf	Public		-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_setBalance	Internal	✓	
RewardToken	Implementation	ERC20, Ownable, Ownership		
	<Constructor>	Public	Payable	ERC20 Ownership
	<Receive Ether>	External	Payable	-
	updateDividendTracker	Public	✓	onlyOwner
	updaterouter	Public	✓	onlyOwner
	excludeFromFees	Public	✓	onlyOwner
	excludeMultipleAccountsFromFees	Public	✓	onlyOwner
	setMarketingWallet	External	✓	onlyOwner
	setTokenRewardsFee	External	✓	onlyOwner
	setLiquiditFee	External	✓	onlyOwner
	setMarketingFee	External	✓	onlyOwner
	setAutomatedMarketMakerPair	Public	✓	onlyOwner
	blacklistAddress	External	✓	onlyOwner
	_setAutomatedMarketMakerPair	Private	✓	
	updateGasForProcessing	Public	✓	onlyOwner
	updateClaimWait	External	✓	onlyOwner
	getClaimWait	External		-
	getTotalDividendsDistributed	External		-
	isExcludedFromFees	Public		-

	withdrawableDividendOf	Public		-
	dividendTokenBalanceOf	Public		-
	excludeFromDividends	External	✓	onlyOwner
	getAccountDividendsInfo	External		-
	getAccountDividendsInfoAtIndex	External		-
	processDividendTracker	External	✓	-
	claim	External	✓	-
	getLastProcessedIndex	External		-
	getNumberOfDividendTokenHolders	External		-
	_transfer	Internal	✓	
	swapAndSendToFee	Private	✓	
	swapAndLiquify	Private	✓	
	swapTokensForEth	Private	✓	
	swapTokensForReward	Private	✓	
	addLiquidity	Private	✓	
	swapAndSendDividends	Private	✓	
RewardDividendTracker	Implementation	Ownable, DividendPayingToken		
	<Constructor>	Public	✓	DividendPayingToken
	<Receive Ether>	External	Payable	-
	_transfer	Internal		
	withdrawDividend	Public		-
	excludeFromDividends	External	✓	onlyOwner
	updateClaimWait	External	✓	onlyOwner
	getLastProcessedIndex	External		-
	getNumberOfTokenHolders	External		-
	getAccount	Public		-
	getAccountAtIndex	Public		-
	canAutoClaim	Private		
	setBalance	External	✓	onlyOwner
	process	Public	✓	-
	processAccount	Public	✓	onlyOwner

Contract Flow



Domain Info

Domain Name	honeyinuu.com
Registry Domain ID	2739511099_DOMAIN_COM-VRSN
Creation Date	2022-11-19T18:42:49Z
Updated Date	2022-11-19T18:45:32Z
Registry Expiry Date	2023-11-19T18:42:49Z
Registrar WHOIS Server	whois.rrpproxy.net
Registrar URL	
Registrar	Key-Systems GmbH
Registrar IANA ID	269

The domain was created 2 days before the creation of the audit. It will expire in 12 months.

There is no public billing information, the creator is protected by the privacy settings.

Summary

There are some functions that can be abused by the owner like stop transactions, manipulate the fees and blacklist addresses. The contract can be converted into a honeypot and prevent users from selling if the owner abuses the admin functions. 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