Cyberscope

Audit Report

Faithcoin

January 2024

Network	ETH
Address	0x4c98e264efb067feaa7f103466bdb115e1cc92dc
Audited by	© cyberscope

Analysis

🔴 Critical 🟠 Medium ⚪ Minor / Informative 🔵 Pass

Severity	Code	Description	Status
🔵	ST	Stops Transactions	Passed
🔵	OTUT	Transfers User's Tokens	Passed
🔵	ELFM	Exceeds Fees Limit	Passed
🔵	MT	Mints Tokens	Passed
🔵	BT	Burns Tokens	Passed
🔵	BC	Blacklists Addresses	Passed

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	IDI	Immutable Declaration Improvement	Unresolved
●	MMU	Misleading Method Usages	Unresolved
●	RED	Redudant Event Declaration	Unresolved
●	RCI	Redundant Check Inefficiency	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	UFC	Unused Functionality Concern	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L22	Potential Locked Ether	Unresolved

Table of Contents

Review

Contract Name	FAITH
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	200 runs
Explorer	https://etherscan.io/address/0x4c98e264efb067feaa7f103466bdb115e1cc92dc
Address	0x4c98e264efb067feaa7f103466bdb115e1cc92dc
Network	ETH
Symbol	FAITH
Decimals	18
Total Supply	1,000,000,000,000
Badge Eligibility	Yes

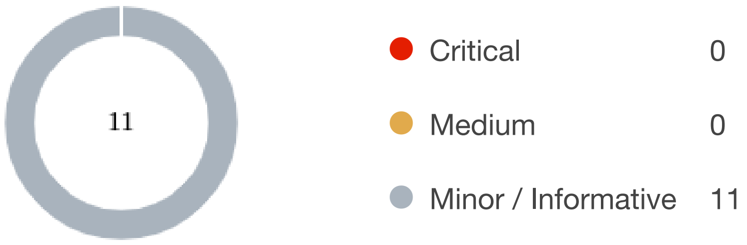
Audit Updates

Initial Audit	11 Jan 2024

Source Files

Filename	SHA256
FAITH.sol	9411a9813a371c00c952887216a53ee86d2ecc7ab5309edbe2d01ded1369cf69

Findings Breakdown



Severity	Unresolved	Acknowledged	Resolved	Other
● Critical	0	0	0	0
● Medium	0	0	0	0
● Minor / Informative	11	0	0	0

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	FAITH.sol#L909
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
lastSync
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

MMU - Misleading Method Usages

Criticality	Minor / Informative
Location	FAITH.sol#L950,1012,1057
Status	Unresolved

Description

The contract includes the `swapbackInfo` , `burnInfo` , `setSwapBackSettings` functions designed to return and set specific state variables. These functions are intended to provide insights into various contract states, like swapback settings and burn mechanism status. However, the returned values are not utilized within the contract's internal logic or external interfaces. This pattern leads to a situation where functions are returning and setting data that, does not influence or interact with the contract's operational processes. The presence of such functions can be misleading, as they suggest a level of interactivity and impact that is not actualized in the contract's execution. This disconnect between the functions' outputs and their practical application or relevance within the broader contract ecosystem raises concerns about the contract's design efficiency and clarity.

```
    function swapbackInfo()
        external
        view
        returns (
            bool _swapbackEnabled,
            uint256 _swapBackValueMin,
            uint256 _swapBackValueMax
        )
    {
        _swapbackEnabled = swapbackEnabled;
        _swapBackValueMin = swapBackValueMin;
        _swapBackValueMax = swapBackValueMax;
    }

    function burnInfo() external view returns (bool
_burnEnabled, uint256 _lastSync) {
        _burnEnabled = burnEnabled;
        _lastSync = lastSync;
    }

    function setSwapBackSettings(
        bool _enabled,
        uint256 _min,
        uint256 _max
    ) external onlyOwner {
        require(
            _min >= 1,
            "Swap amount cannot be lower than 0.01% total
supply."
        );
        require(_max >= _min, "maximum amount cant be higher
than minimum");
        swapbackEnabled = _enabled;
        swapBackValueMin = (totalSupply() * _min) / 10000;
        swapBackValueMax = (totalSupply() * _max) / 10000;
        emit SwapbackSettingsUpdated(_enabled, _min, _max);
    }
```

Recommendation

It is recommended to conduct a comprehensive review of the contract to determine the necessity and intended use of these functions. If the purpose of these functions is solely to provide information that might be used, it should be explicitly documented to clarify their role and avoid any misunderstanding. On the other hand, if these functions are not contributing to the contract's core functionality, considering their removal or modification could be beneficial. This approach will streamline the contract, ensuring that each component serves a distinct and practical purpose. Simplifying the contract by removing or repurposing non-essential functions can lead to improved clarity, making the contract more accessible and maintainable for developers, auditors, and users. This refinement process will contribute to the overall robustness and transparency of the contract.

RED - Redudant Event Declaration

Criticality	Minor / Informative
Location	FAITH.sol#L885
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 event `UpdateUniswapV2Router` is declared and not being used in the contract. As a result, it is redundant.

```
    event UpdateUniswapV2Router(
        address indexed newAddress,
        address indexed oldAddress
    );
```

Recommendation

The team is advised to take these segments into consideration 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.

RCI - Redundant Check Inefficiency

Criticality	Minor / Informative
Location	FAITH.sol#L1163
Status	Unresolved

Description

The contract is structured with nested if statements that include checks to determine the status of various conditions before executing certain operations. Specifically, within the outer if statement, the condition checks whether both from and to are not equal to owner, among other conditions. Subsequently, within an inner if statement, there is a repeated check to ascertain if `to != owner()` . This repetition is redundant because the condition `to != owner()` has already been validated in the preceding outer if statement. As a result, the second check for `to != owner()` within the inner if statement is unnecessary and does not contribute any additional logic or security to the contract. This redundancy can lead to inefficiencies in the contract's execution and may cause confusion or misinterpretation of the contract's intended logic.

```
        if (limitsInEffect) {
            if (
                from != owner() &&
                to != owner() &&
                to != address(0) &&
                to != address(0xdead) &&
                !swapping
            ) {

                if (transferDelayEnabled) {
                    if (
                        to != owner() &&
                        to != address(dexRouter) &&
                        to != address(dexPair)
                    )
            }
```

Recommendation

It is recommended to remove the second if statement that checks `to != owner()` within the inner if block. Since this condition is already evaluated in the outer if statement, its presence in the inner block is superfluous and does not alter the outcome of the conditional checks. Eliminating this redundant check will streamline the contract's logic, making it more efficient and easier to understand. This simplification will not only enhance the readability of the contract but also reduce the potential for errors or misunderstandings related to the contract's intended behavior.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	FAITH.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.

UFC - Unused Functionality Concern

Criticality	Minor / Informative
Location	FAITH.sol#L870,1023,1207
Status	Unresolved

Description

The contract is designed with a structure that includes variables and functions related to trading activities, specifically the `tradingOn` boolean variable and the `enableTrading` and `swapTokensForEth` functions. These elements are integral to the contract's intended trading functionality. The `tradingOn` variable is a public boolean set to `false` by default, indicating whether trading is enabled or not. The `swapTokensForEth` function is a private method intended for swapping tokens for Ethereum (ETH), crucial for executing trades. However, the `tradingOn` variable along with the `swapTokensForEth` function, are never actually invoked within the contract. This omission results in a significant functionality gap, as the mechanisms to initiate and execute trading processes, despite being coded, are not utilized in practice. This raises concerns about the contract's operational effectiveness and its alignment with the intended trading functionalities.

```
   bool public tradingOn = false;

    function enableTrading() external onlyOwner {
        tradingOn = true;
        swapbackEnabled = true;
        emit TradingEnabled(block.timestamp);
    }

    function swapTokensForEth(uint256 tokenAmount) private {
        // generate the uniswap pair path of token -> weth
        address[] memory path = new address[](2);
        path[0] = address(this);
        path[1] = dexRouter.WETH();

        _approve(address(this), address(dexRouter),
tokenAmount);

        // make the swap

dexRouter.swapExactTokensForETHSupportingFeeOnTransferTokens(
            tokenAmount,
            0, // accept any amount of ETH
            path,
            address(this),
            block.timestamp
        );
    }
```

Recommendation

It is recommended to thoroughly review and reconsider the intended functionality of the contract. If the purpose is to include trading capabilities, then the contract should be amended to utilize the `enableTrading` function and the `swapTokensForEth` function appropriately. This could involve integrating these functions into the contract's workflow where trading and swap activities are expected to occur. Ensuring that these functions are called at the appropriate points will activate the trading features as intended and align the contract's functionality with its design. This approach will enhance the contract's reliability and effectiveness in facilitating trading activities.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	FAITH.sol#L854,857,873
Status	Unresolved

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
address private newOwner =
0x4D32DA5EbdF32561ce802bBFB43dE6Ca40363d3D
bool private swapping
bool private burnEnabled = true
```

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	FAITH.sol#L800,994,1058,1059,1060
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);
address _target
bool _enabled
uint256 _min
uint256 _max
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.

Find more information on the Solidity documentation https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	FAITH.sol#L620,1206
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function _burn(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: burn from the zero
address");

        _beforeTokenTransfer(account, address(0), amount);

        uint256 accountBalance = _balances[account];
...
        }
        _totalSupply -= amount;

        emit Transfer(account, address(0), amount);

        _afterTokenTransfer(account, address(0), amount);
    }

...
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	FAITH.sol#L921
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 = 1000000000000 * 10 ** decimals()
```

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.

L22 - Potential Locked Ether

Criticality	Minor / Informative
Location	FAITH.sol#L1017
Status	Unresolved

Description

The contract contains Ether that has been placed into a Solidity contract and is unable to be transferred. Thus, it is impossible to access the locked Ether. This may produce a financial loss for the users that have called the payable method.

```
receive() external payable {}
```

Recommendation

The team is advised to either remove the payable method or add a withdraw functionality. it is important to carefully consider the risks and potential issues associated with locked Ether.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
SafeMath	Library			
	tryAdd	Internal		
	trySub	Internal		
	tryMul	Internal		
	tryDiv	Internal		
	tryMod	Internal		
	add	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	mod	Internal		
	sub	Internal		
	div	Internal		
	mod	Internal		
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-

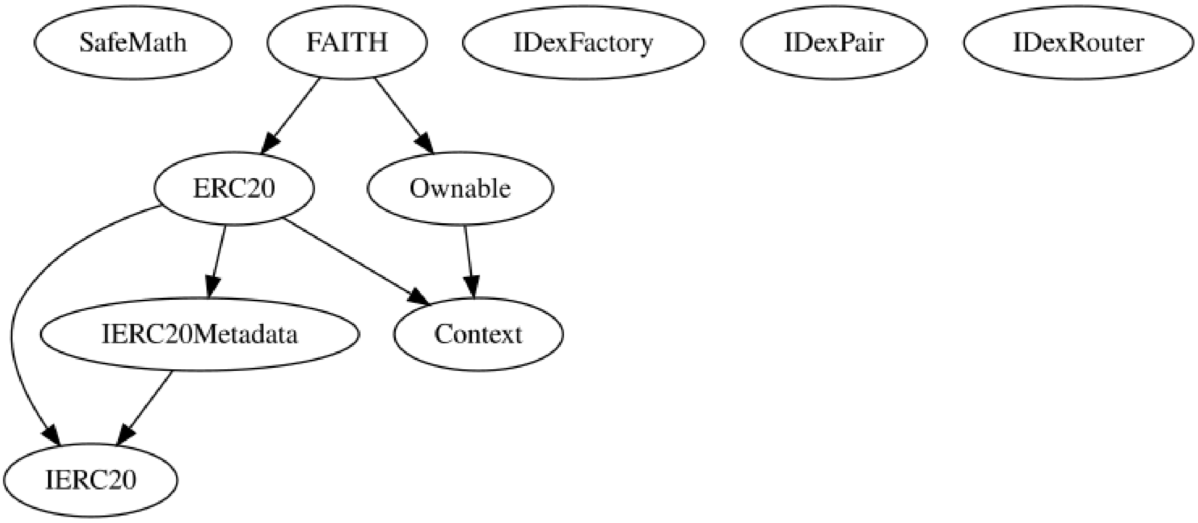
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public	✓	-
	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	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
IDexFactory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-

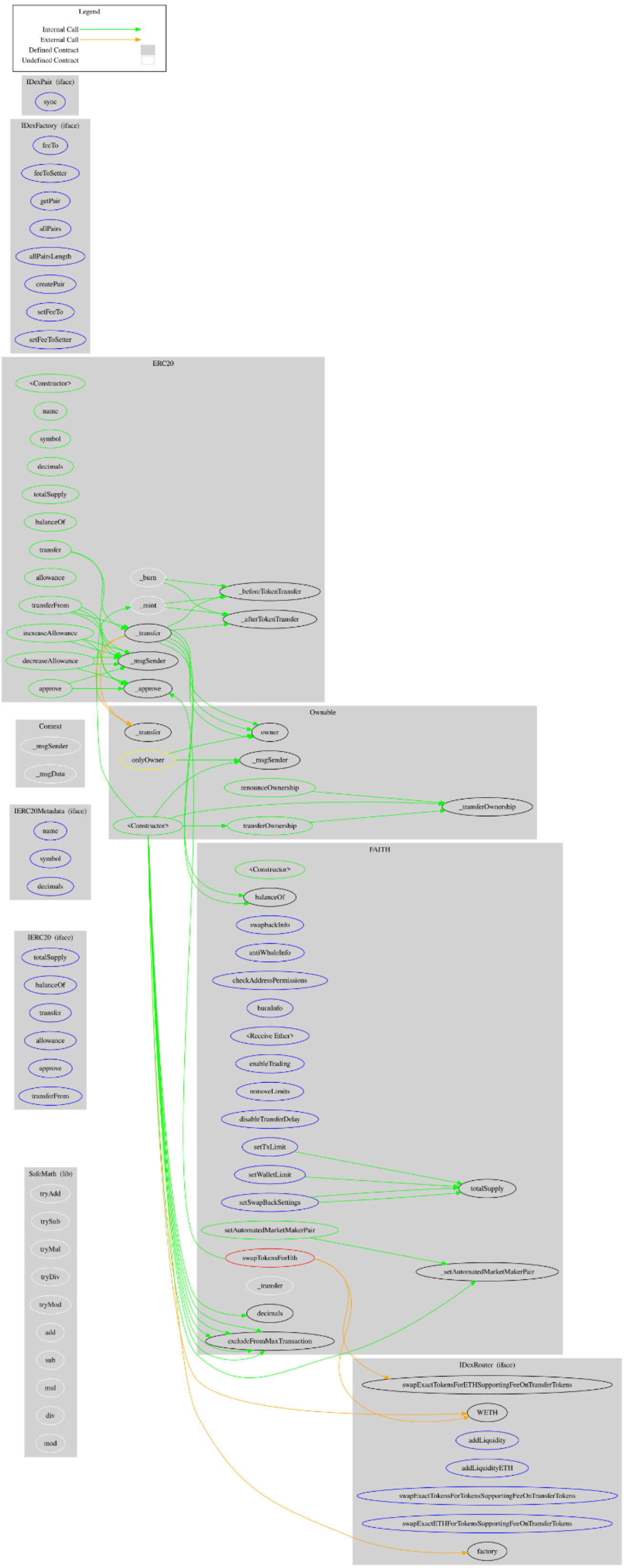
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-
IDexPair	Interface			
	sync	External	✓	-
IDexRouter	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
FAITH	Implementation	ERC20, Ownable		
		Public	✓	ERC20
	decimals	Public		-
	swapbackInfo	External		-
	antiWhaleInfo	External		-

	checkAddressPermissions	External		-
	burnInfo	External		-
		External	Payable	-
	enableTrading	External	✓	onlyOwner
	removeLimits	External	✓	onlyOwner
	disableTransferDelay	External	✓	onlyOwner
	setSwapBackSettings	External	✓	onlyOwner
	setTxLimit	External	✓	onlyOwner
	setWalletLimit	External	✓	onlyOwner
	excludeFromMaxTransaction	Public	✓	onlyOwner
	setAutomatedMarketMakerPair	Public	✓	onlyOwner
	_setAutomatedMarketMakerPair	Private	✓	
	_transfer	Internal	✓	
	swapTokensForEth	Private	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IDexPair (iface)
sync
IDexFactory (iface)
feeTo
feeToSetter
getPair
allPairs
allPairsLength
createPair
setFeeTo
setFeeToSetter
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
transfer
allowance
transferFrom
increaseAllowance
decreaseAllowance
approve
_burn
_mint
_transfer
_msgSender
_approve
_beforeTokenTransfer
_afterTokenTransfer
Context
_msgSender
_msgData
Ownable
_transfer
onlyOwner
owner
_msgSender
renounceOwnership
_transferOwnership
<Constructor>
transferOwnership
IERC20Metadata (iface)
name
symbol
decimals
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
FAITH
<Constructor>
balanceOf
swapbackInfo
antiWhaleInfo
checkAddressPermissions
burnInfo
<Receive Ether>
enableTrading
removeLimits
disableTransferDelay
setTxLimit
setWalletLimit
setSwapBackSettings
totalSupply
setAutomatedMarketMakerPair
swapTokensForEth
_setAutomatedMarketMakerPair
_transfer
decimals
excludeFromMaxTransaction
SafeMath (lib)
tryAdd
trySub
tryMul
tryDiv
tryMod
add
sub
mul
div
mod
IDexRouter (iface)
swapExactTokensForETHSupportingFeeOnTransferTokens
WETH
addLiquidity
addLiquidityETH
swapExactTokensForTokensSupportingFeeOnTransferTokens
swapExactETHForTokensSupportingFeeOnTransferTokens
factory

Summary

Faithcoin contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. Faithcoin is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues. The contract Owner can access some admin functions that can not be used in a malicious way to disturb the users' transactions.

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