Cyberscope

Audit Report

Flack Exchange

January 2024

Network ETH

Address 0xF31120603a27A16314efFc37a3F32A42028310af

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
	CRO	Code Readability Optimization	Unresolved
	IDI	Immutable Declaration Improvement	Unresolved
	MCM	Misleading Comment Messages	Unresolved
	MEE	Missing Events Emission	Unresolved
	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
	PTRP	Potential Transfer Revert Propagation	Unresolved
	PVC	Price Volatility Concern	Unresolved
	RRS	Redundant Require Statement	Unresolved
	RSW	Redundant Storage Writes	Unresolved
	L02	State Variables could be Declared Constant	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L07	Missing Events Arithmetic	Unresolved
	L14	Uninitialized Variables in Local Scope	Unresolved
	L16	Validate Variable Setters	Unresolved

●	L19	Stable Compiler Version	Unresolved

Table of Contents

Review

Contract Name	Flack
Compiler Version	v0.8.22+commit.4fc1097e
Optimization	200 runs
Explorer	https://etherscan.io/address/0xf31120603a27a16314effc37a3f32a42028310af
Address	0xf31120603a27a16314effc37a3f32a42028310af
Network	ETH
Symbol	FLACK
Decimals	18
Total Supply	1,000,000
Badge Eligibility	Must Fix Criticals

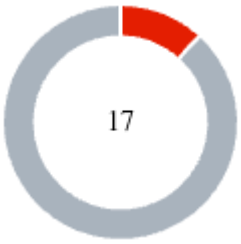
Audit Updates

Initial Audit	21 Jan 2024

Source Files

Filename	SHA256
Flack.sol	21f2a5f1e8e4b9bed65e82ed2f62e8417fa781ef26bdcf6df7eb3c7ebbdf6cc8

Findings Breakdown



Critical	2
Medium	0
Minor / Informative	15

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	2	0	0	0
Medium	0	0	0	0
Minor / Informative	15	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	Flack.sol#L307,309
Status	Unresolved

Description

The contract owner has the authority to stop the transactions for all users excluding the authorized addresses. The owner may take advantage of it by setting the `maxTransaction` to zero.

```
require(amount <= maxTransaction, "Exceeds maxTxAmount");
```

The contract owner has the authority to stop the buys and transfers for all users excluding the authorized addresses. The owner may take advantage of it by setting the `maxWallet` to zero.

```
if(!isMarketPair[recipient]) {
    require(balanceOf(recipient).add(amount) <= maxWallet, "Exceeds
maxWallet");
}
```

The contract owner has the authority to stop the sales for all users excluding the authorized addresses. The owner may take advantage of it by setting the `sellFee` over 100%. Additionally, the issues, which are described in details, in sections PLPI, PTRP, and PVC can also result in the suspension of sales. As a result, the contract may operate as a honeypot.

```
feeAmount = amount.mul(sellFee).div(feeDenominator);
...
return amount.sub(feeAmount);
```

Recommendation

The contract could embody a check for not allowing setting the `maxTransaction` and `maxWallet` less than a reasonable amount. A suggested implementation could check that the minimum amount should be more than a fixed percentage of the total supply. 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.

ELFM - Exceeds Fees Limit

Criticality	Critical
Location	Flack.sol#L429
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 `setFee` function with a high percentage value.

```
function setFee(uint _buySide, uint _sellSide) external onlyOwner {
    buyFee = _buySide;
    sellFee = _sellSide;
}
```

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.

CRO - Code Readability Optimization

Criticality	Minor / Informative
Location	Flack.sol#L347
Status	Unresolved

Description

The `checkSwapFee` function, designed to determine whether fees should be applied to a transaction, currently features a more complex implementation than necessary. The function checks various conditions and returns true only if either the sender or the recipient is in the `_chargePair` mapping. In all other cases, it returns false. The logic can be simplified by directly returning `_chargePair[sender] || _chargePair[recipient]`.

```
function checkSwapFee(address sender, address recipient) internal view
returns (bool) {
    if(_chargePair[sender] || _chargePair[recipient]) {
        return true;
    }
    else if (isMarketPair[sender] || isMarketPair[recipient]) {
        return false;
    }
    else {
        return false;
    }
}
```

Recommendation

The is advised to simplify the logic in the `checkSwapFee` function by directly returning `_chargePair[sender] || _chargePair[recipient]`. This streamlines the code, making it more concise and easier to understand. The simplified version maintains the intended behavior of the function while reducing unnecessary complexity.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	Flack.sol#L220
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.

```
dexPair
```

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.

MCM - Misleading Comment Messages

Criticality	Minor / Informative
Location	Flack.sol#L177
Status	Unresolved

Description

The contract is using misleading comment messages. These comment messages do not accurately reflect the actual implementation, making it difficult to understand the source code. As a result, the users will not comprehend the source code's actual implementation.

```
// Anti Whale Mechanism with 1% of Max Bag and Transaction
uint256 public maxTransaction =  _totalSupply.mul(5).div(100);
uint256 public maxWallet = _totalSupply.mul(3).div(100);
```

Recommendation

The team is advised to carefully review the comment in order to reflect the actual implementation. To improve code readability, the team should use more specific and descriptive comment messages.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	Flack.sol#L435,441,453,457,463
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.

```
LimitsActive = false;
_chargePair[_adr] = _status;
marketingWallet = _newWallet;
developerWallet = _newWallet;
swapEnabled = _swapenabled;
```

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.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	Flack.sol#L397
Status	Unresolved

Description

The contract operates under the assumption that liquidity is consistently provided to the pair between the contract's token and the native currency. However, there is a possibility that liquidity is provided to a different pair. This inadequacy in liquidity provision in the main pair could expose the contract to risks. Specifically, during eligible transactions, where the contract attempts to swap tokens with the main pair, a failure may occur if liquidity has been added to a pair other than the primary one. Consequently, transactions triggering the swap functionality will result in a revert.

```
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), // The contract
        block.timestamp
    );

    emit SwapTokensForETH(tokenAmount, path);
}
```

Recommendation

The team is advised to implement a runtime mechanism to check if the pair has adequate liquidity provisions. This feature allows the contract to omit token swaps if the pair does not have adequate liquidity provisions, significantly minimizing the risk of potential failures.

Furthermore, the team could ensure the contract has the capability to switch its active pair in case liquidity is added to another pair.

Additionally, the contract could be designed to tolerate potential reverts from the swap functionality, especially when it is a part of the main transfer flow. This can be achieved by executing the contract's token swaps in a non-reversible manner, thereby ensuring a more resilient and predictable operation.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	Flack.sol#L393
Status	Unresolved

Description

The contract sends funds to a `marketingWallet` as part of the transfer flow. This address can either be a wallet address or a contract. If the address belongs to a contract then it may revert from incoming payment. As a result, the error will propagate to the token's contract and revert the transfer.

```
payable(marketingWallet).transfer(amountReceived);
```

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

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	Flack.sol#L467
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `swapThreshold` 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 setSwapThreshold(uint _threshold)
    external
    onlyOwner
{
    swapThreshold = _threshold;
}
```

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.

RRS - Redundant Require Statement

Criticality	Minor / Informative
Location	Flack.sol#L93
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.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	Flack.sol#L435,441
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.

```
LimitsActive = false;
_chargePair[_adr] = _status;
```

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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	Flack.sol#L168,169,170,187
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.

```
string _name = "Flack Exchange"
string _symbol = "FLACK"
uint8 _decimals = 18
uint256 feeDenominator = 100
```

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	Flack.sol#L142,165,194,360,423,429,440,452,456,460,467
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);
mapping (address => bool) public _chargePair
bool public LimitsActive = true

...
```

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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	Flack.sol#L430,445,449,471
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.

```
buyFee = _buySide
maxWallet = newLimit
maxTransaction = newLimit
swapThreshold = _threshold
```

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.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	Flack.sol#L362
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.

```
uint feeAmount
```

Recommendation

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

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	Flack.sol#L425,453,457
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.

```
(bool success, ) =
address(_token).call(abi.encodeWithSignature('transfer(address,uint256)',
developerWallet, _amount))
marketingWallet = _newWallet
developerWallet = _newWallet
```

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	Flack.sol#L22
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 ^0.8.22;
```

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.

Functions Analysis

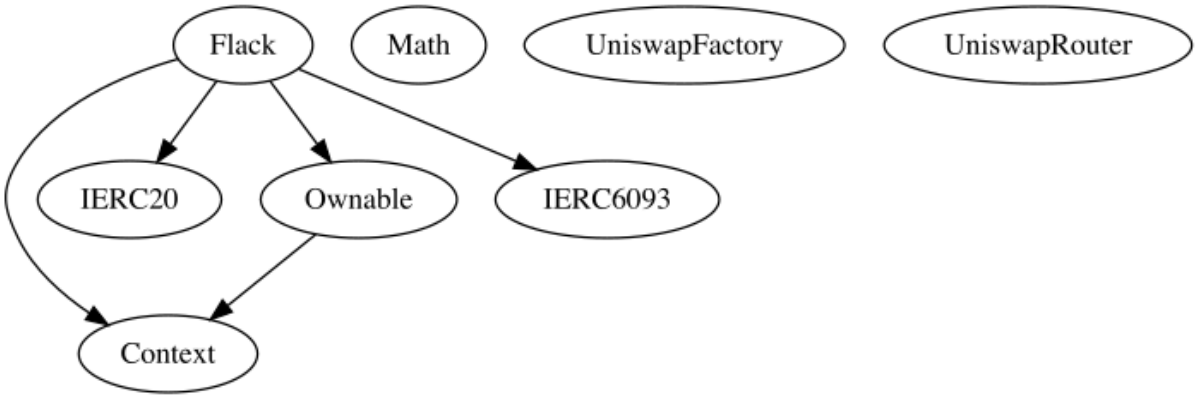
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
	_msgSender	Internal		
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_setOwner	Private	✓	

Math	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
UniswapFactory	Interface			
	createPair	External	✓	-
UniswapRouter	Interface			
	factory	External		-
	WETH	External		-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
IERC6093	Interface			
Flack	Implementation	Context, IERC20, Ownable, IERC6093		
		Public	✓	-
	name	Public		-
	symbol	Public		-

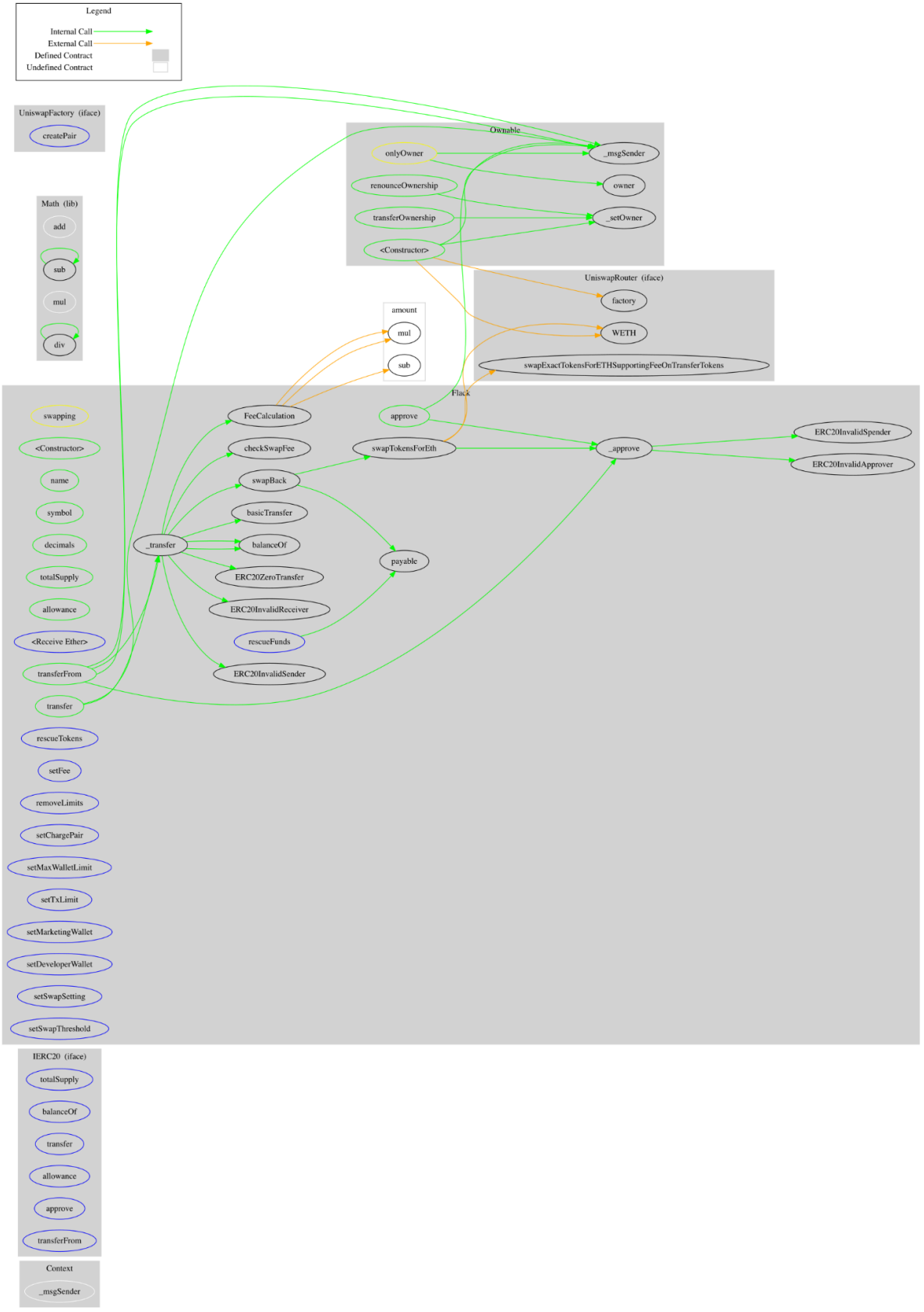
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	allowance	Public		-
	approve	Public	✓	-
	_approve	Private	✓	
		External	Payable	-
	transfer	Public	✓	-
	transferFrom	Public	✓	-
	_transfer	Private	✓	
	basicTransfer	Internal	✓	
	checkSwapFee	Internal		
	FeeCalculation	Internal	✓	
	swapBack	Internal	✓	swapping
	swapTokensForEth	Private	✓	
	rescueFunds	External	✓	-
	rescueTokens	External	✓	-
	setFee	External	✓	onlyOwner
	removeLimits	External	✓	onlyOwner
	setChargePair	External	✓	onlyOwner
	setMaxWalletLimit	External	✓	onlyOwner
	setTxLimit	External	✓	onlyOwner
	setMarketingWallet	External	✓	onlyOwner

	setDeveloperWallet	External	✓	onlyOwner
	setSwapSetting	External	✓	onlyOwner
	setSwapThreshold	External	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
UniswapFactory (iface)
createPair
Math (lib)
add
sub
mul
div
Ownable
onlyOwner
_msgSender
renounceOwnership
owner
transferOwnership
_setOwner
<Constructor>
UniswapRouter (iface)
factory
WETH
swapExactTokensForETHSupportingFeeOnTransferTokens
amount
mul
sub
Flack
swapping
<Constructor>
name
symbol
decimals
totalSupply
allowance
<Receive Ether>
transferFrom
transfer
rescueTokens
setFee
removeLimits
setChargePair
setMaxWalletLimit
setTxLimit
setMarketingWallet
setDeveloperWallet
setSwapSetting
setSwapThreshold
FeeCalculation
checkSwapFee
swapBack
basicTransfer
_transfer
balanceOf
ERC20ZeroTransfer
ERC20InvalidReceiver
rescueFunds
ERC20InvalidSender
payable
approve
swapTokensForEth
_approve
ERC20InvalidSpender
ERC20InvalidApprover
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Context
_msgSender

Summary

Flack Exchange 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 stopping transactions and manipulating the fees. 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