Cyberscope

Audit Report
FEELR

Jule 2023

Network	ETH
Address	0x332679057B9E8b541993d658069543f7AdFDab93
Audited by	© cyberscope

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Acknowledged
●	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
●	SFV	Swap Functionality Vulnerability	Acknowledged
●	FSA	Fixed Swap Address	Unresolved
●	CR	Code Repetition	Unresolved
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Analysis **1**
Diagnostics **2**
Table of Contents **3**
Review **5**
- Audit Updates 5
- Source Files 5

Findings Breakdown **6**
- ST - Stops Transactions 7
  - Description 7
  - Recommendation 7
- SFV - Swap Functionality Vulnerability 8
  - Description 8
  - Recommendation 9
  - Team Update 9
- FSA - Fixed Swap Address 10
  - Description 10
  - Recommendation 10
- CR - Code Repetition 11
  - Description 11
  - Recommendation 11
- PTRP - Potential Transfer Revert Propagation 12
  - Description 12
  - Recommendation 12
- RSW - Redundant Storage Writes 13
  - Description 13
  - Recommendation 13
- PVC - Price Volatility Concern 15
  - Description 15
  - Recommendation 15
- RSML - Redundant SafeMath Library 16
  - Description 16
  - Recommendation 16
- L02 - State Variables could be Declared Constant 17
  - Description 17
  - Recommendation 17
- L04 - Conformance to Solidity Naming Conventions 18
  - Description 18

Recommendation 18
L05 - Unused State Variable 20
Description 20
Recommendation 20
L07 - Missing Events Arithmetic 21
Description 21
Recommendation 21
L16 - Validate Variable Setters 22
Description 22
Recommendation 22
L19 - Stable Compiler Version 23
Description 23
Recommendation 23
Functions Analysis 24
Inheritance Graph 28
Flow Graph 29
Summary 30
Disclaimer 31
About Cyberscope 32

Review

Contract Name	FeelrToken
Compiler Version	v0.8.10+commit.fc410830
Optimization	200 runs
Explorer	https://etherscan.io/address/0x332679057b9e8b541993d658069543f7adfdab93
Address	0x332679057b9e8b541993d658069543f7adfdab93
Network	ETH
Symbol	$Feelr
Decimals	9
Total Supply	1.000.000.000.000

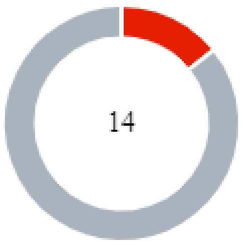
Audit Updates

Initial Audit	02 Jul 2023 https://github.com/cyberscope-io/audits/blob/main/feelr/v1/audit.pdf
Corrected Phase 2	04 Jul 2023

Source Files

Filename	SHA256
FeelrToken.sol	b64550e175e40b1ba2ed7690c33bbe232391f4614c915b00bfcf4f7b382d534b

Findings Breakdown



●	Critical	2
●	Medium	0
●	Minor / Informative	12

Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	0	2	0	0
●	Medium	0	0	0	0
●	Minor / Informative	12	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	FeelrToken.sol#L379
Status	Acknowledged

Description

The transactions are initially disabled 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 (!isExcludedFromMaxTxn[from] && !isExcludedFromMaxTxn[to]) {
  // trading disable till launch
  if (!trading) {
      require(
          dexPair != from && dexPair != to,
          "$Feelr: trading is disable"
      );
  }
}
```

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.

SFV - Swap Functionality Vulnerability

Criticality	Critical
Location	contracts/GameCraft.sol#L468
Status	Acknowledged

Description

As part of the transfer flow, the contract has an implementation of the swap functionality. However, it lacks a mutex in its implementation. During the swapping process, the `transfer` function will be called internally. As a consequence of the second swap lacking a mutex, the contract can fall into an infinite loop, continuously executing the swap operation. This continuous execution consumes all the available gas and eventually reaches the limit, causing the transaction to fail and revert.

```
function distributeAndLiquify(address from, address to) private {
    // is the token balance of this contract address over the min
number of
    // tokens that we need to initiate a swap + liquidity lock?
    // also, don't get caught in a circular liquidity event.
    // also, don't swap & liquify if sender is Dex pair.
    uint256 contractTokenBalance = balanceOf(address(this));

    bool shouldSell = contractTokenBalance >= minTokenToSwap;

    if (
        shouldSell &&
        from != dexPair &&
        autoSwapStatus &&
        !(from == address(this) && to == dexPair) // swap 1 time
    ) {
        // approve contract
        _approve(address(this), address(dexRouter), minTokenToSwap);

        // now is to lock into liquidty pool
        Utils.swapTokensForEth(address(dexRouter), minTokenToSwap);

        uint256 ethForMarketing = address(this).balance;

        // sending Eth to Marketing wallet
        if (ethForMarketing > 0)
            payable(marketingWallet).transfer(ethForMarketing);
    }
}
```

Recommendation

The team is advised to take these segments into consideration and introduce a mutex. This way the contract will avoid entering an infinite loop, causing the transaction to revert.

Team Update

"The Swap and distribute function doesn't have a function that can stop an infinite loop but that loop can only happen in case of a bot and even if does happen the transaction wont go through. For a regular investor this doesn't bother him."

FSA - Fixed Swap Address

Criticality	Minor / Informative
Location	FeelrToken.sol#L155
Status	Unresolved

Description

The swap address is assigned once and it can not be changed. It is a common practice in decentralized exchanges to create new swap versions. A contract that cannot change the swap address may not be able to catch up to the upgrade. As a result, the contract will not be able to migrate to a new liquidity pool pair or decentralized exchange.

```
constructor() {
    _balances[owner()] = _totalSupply;
    marketingWallet =
address(0xAd8F6242c3965296ce1871668A6810DbFcB0f632);

    IDexRouter _dexRouter = IDexRouter(
        0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D
    );
    // Create a dex pair for this new ERC20
    address _dexPair = IDexFactory(_dexRouter.factory()).createPair(
        address(this),
        _dexRouter.WETH()
    );
    dexPair = _dexPair;

    // set the rest of the contract variables
    dexRouter = _dexRouter;
```

Recommendation

The team is advised to add the ability to change the pair and router address in order to cover potential liquidity pool migrations. It would be better to support multiple pair addresses so the token will be able to have the same behavior in all the decentralized liquidity pairs.

CR - Code Repetition

Criticality	Minor / Informative
Location	FeelrToken.sol#L433,443,453
Status	Unresolved

Description

The contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

```
_balances[sender] = _balances[sender].sub(
    amount,
    "$Feelr: insufficient balance"
);
_balances[recipient] = _balances[recipient].add(tTransferAmount);
emit Transfer(sender, recipient, tTransferAmount);
```

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help reduce the complexity and size of the contract. For instance, the contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	FeelrToken.sol#L493
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.

```
if (ethForMarketing > 0)
        payable(marketingWallet).transfer(ethForMarketing);
```

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.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	FeelrToken.sol#L282,289,296,332,336,340
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 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.

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

function includeOrExcludeFromMaxTxn(
    address account,
    bool value
) external onlyOwner {
    isExcludedFromMaxTxn[account] = value;
}

function includeOrExcludeFromMaxHolding(
    address account,
    bool value
) external onlyOwner {
    isExcludedFromMaxHolding[account] = value;
}

...
```

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.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	FeelrToken.sol#L468
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `minTokenToSwap` 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 distributeAndLiquify(address from, address to) private {
        // is the token balance of this contract address over the min
number of
        // tokens that we need to initiate a swap + liquidity lock?
        // also, don't get caught in a circular liquidity event.
        // also, don't swap & liquify if sender is Dex pair.
        uint256 contractTokenBalance = balanceOf(address(this));

        bool shouldSell = contractTokenBalance >= minTokenToSwap;
```

Recommendation

The contract could ensure that it will not sell more than a reasonable amount of tokens in a single transaction. A suggested implementation could check that the maximum amount should be less than a fixed percentage of the total supply. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	FeelrToken.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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	FeelrToken.sol#L129,130,131,132,142,143,144
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 private _name = "Feelr"
string private _symbol = "$Feelr"
uint8 private _decimals = 9
uint256 private _totalSupply = 1_000_000_000_000 * 1e9
uint256 public botFee = 990
uint256 public percentDivider = 1000
uint256 public snipingTime = 60 seconds
```

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	FeelrToken.sol#L48,303,308,316,324,332,336,340,352
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);
uint256 _amount
uint256 _marketingFee
bool _value
address _marketingWallet
address _receiver
```

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	FeelrToken.sol#L134,135
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.

```
address private constant DEAD = address(0xdead)
address private constant ZERO = address(0)
```

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	FeelrToken.sol#L305,313,317,325
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.

```
minTokenToSwap = _amount
maxHoldLimit = _amount
marketingFeeOnBuying = _marketingFee
marketingFeeOnSelling = _marketingFee
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	FeelrToken.sol#L341,353
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
payable(_receiver).transfer(address(this).balance)
```

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	FeelrToken.sol#L2
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 ^0.8.10;
```

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

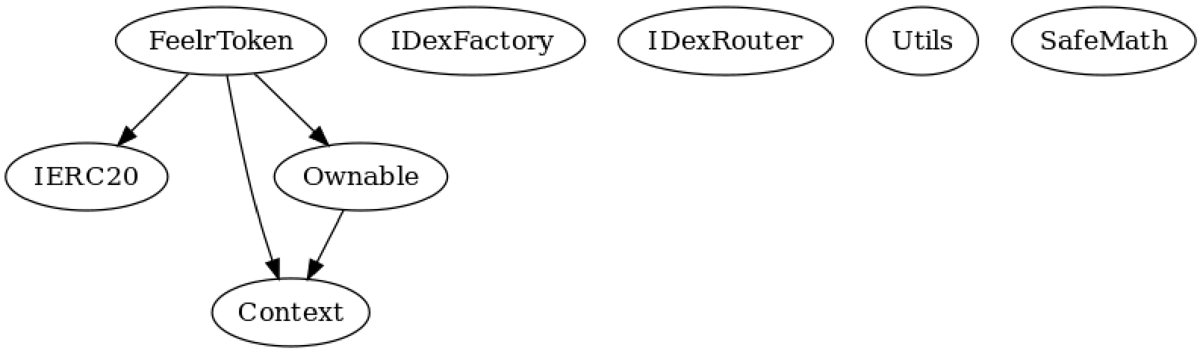
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IDexFactory	Interface			
	createPair	External	✓	-
IDexRouter	Interface			
	factory	External		-
	WETH	External		-
	addLiquidityETH	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
Context	Implementation			

	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
FeelrToken	Implementation	Context, IERC20, Ownable		
		Public	✓	-
		External	Payable	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-

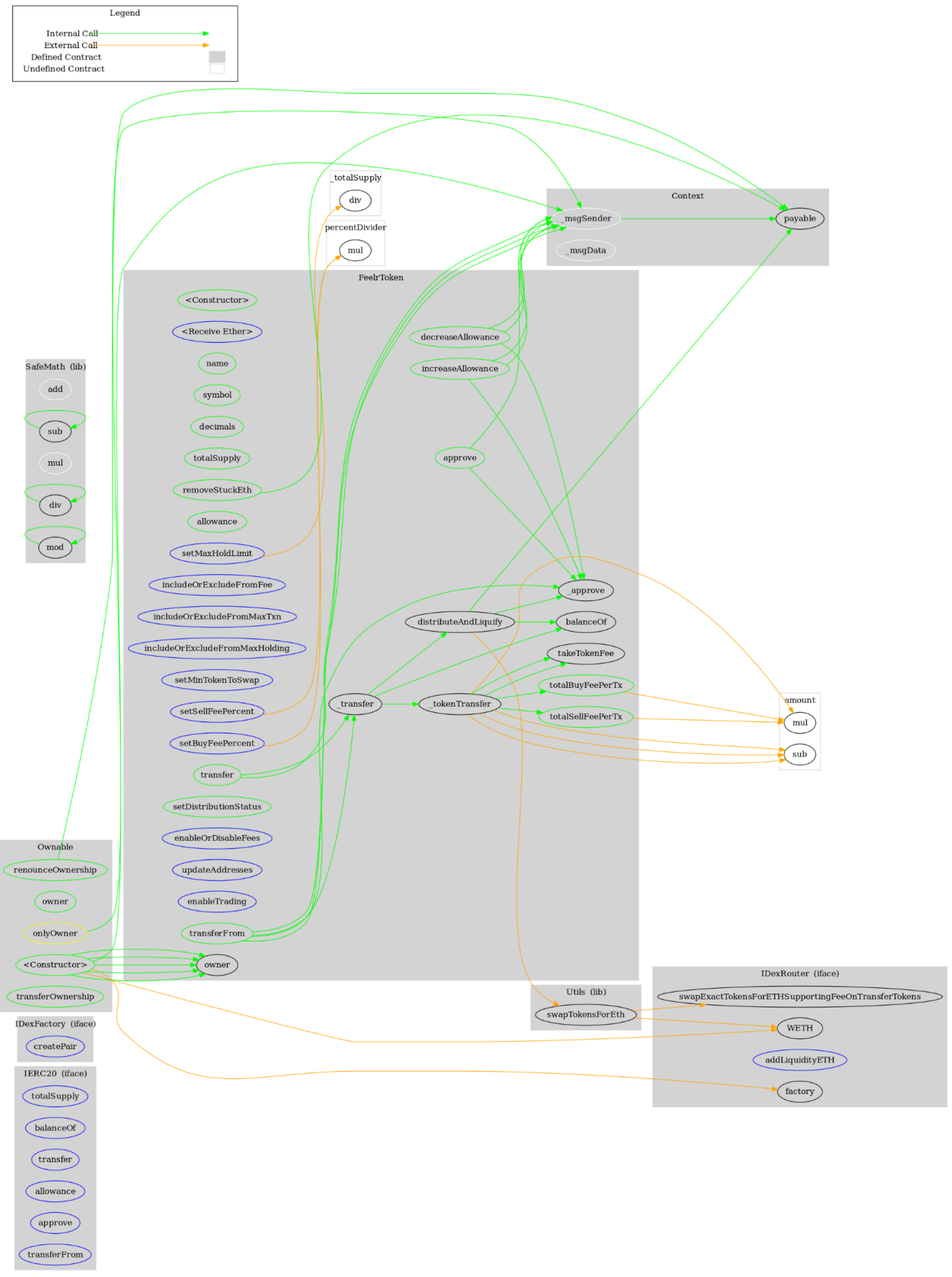
	includeOrExcludeFromFee	External	✓	onlyOwner
	includeOrExcludeFromMaxTxn	External	✓	onlyOwner
	includeOrExcludeFromMaxHolding	External	✓	onlyOwner
	setMinTokenToSwap	External	✓	onlyOwner
	setMaxHoldLimit	External	✓	onlyOwner
	setBuyFeePercent	External	✓	onlyOwner
	setSellFeePercent	External	✓	onlyOwner
	setDistributionStatus	Public	✓	onlyOwner
	enableOrDisableFees	External	✓	onlyOwner
	updateAddresses	External	✓	onlyOwner
	enableTrading	External	✓	onlyOwner
	removeStuckEth	Public	✓	onlyOwner
	totalBuyFeePerTx	Public		-
	totalSellFeePerTx	Public		-
	_approve	Private	✓	
	_transfer	Private	✓	
	_tokenTransfer	Private	✓	
	takeTokenFee	Private	✓	
	distributeAndLiquify	Private	✓	
Utils	Library			
	swapTokensForEth	Internal	✓	

SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
	mod	Internal		
	mod	Internal		

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
_totalSupply
div
percentDivider
mul
Context
_msgSender
payable
_msgData
FeelrToken
<Constructor>
<Receive Ether>
name
symbol
decimals
totalSupply
removeStuckEth
allowance
setMaxHoldLimit
includeOrExcludeFromFee
includeOrExcludeFromMaxTxn
includeOrExcludeFromMaxHolding
setMinTokenToSwap
setSellFeePercent
setBuyFeePercent
transfer
setDistributionStatus
enableOrDisableFees
updateAddresses
enableTrading
transferFrom
owner
decreaseAllowance
increaseAllowance
approve
_approve
distributeAndLiquify
balanceOf
takeTokenFee
totalBuyFeePerTx
transfer
_tokenTransfer
totalSellFeePerTx
amount
mul
sub
SafeMath (lib)
add
sub
mul
div
mod
Ownable
renounceOwnership
owner
onlyOwner
<Constructor>
transferOwnership
IDexFactory (iface)
createPair
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Utils (lib)
swapTokensForEth
IDexRouter (iface)
swapExactTokensForETHSupportingFeeOnTransferTokens
WETH
addLiquidityETH
factory

Summary

FEELR 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. 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 15% fee. The contract implements an antibot mechanism that applies 99% fees on the first 60 seconds after the contract launches.

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