Cyberscope

Audit Report

Feed Coin

February 2023

Type	BEP20
Network	BSC
Address	0xbc412024c1F7112c8C3DE4F9Da401b23ca69Ffa7
Audited by	© cyberscope

Table of Contents

Review

Contract Name	Feedcoin
Compiler Version	v0.8.7+commit.e28d00a7
Optimization	200 runs
Explorer	https://bscscan.com/address/0xbc412024c1f7112c8c3de4f9da401b23ca69ffa7
Address	0xbc412024c1f7112c8c3de4f9da401b23ca69ffa7
Network	BSC
Symbol	Feed
Decimals	18
Total Supply	10,000,000,000

Audit Updates

Initial Audit	02 Feb 2023

Source Files

Filename	SHA256
Feedcoin.sol	dfc024f9142f8c9df9ed25901e699a4e57fd7d3ecfff2ecde77b8342d08ba078

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	OCTD	Transfers Contract's Tokens	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	ULTW	Transfers Liquidity to Team Wallet	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L12	Using Variables before Declaration	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L18	Multiple Pragma Directives	Unresolved
●	L19	Stable Compiler Version	Unresolved

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	Feedcoin.sol#L384,385,386,388,389,390,519,1050,1095
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.

```
es32 private immutable _CACHED_DOMAIN_SEPARATOR;


t256 private immutable _CACHED_CHAIN_ID;


...


es32 private immutable _TYPE_HASH;


ction DOMAIN_SEPARATOR() external view returns (bytes32);
}

...
```

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	Feedcoin.sol#L34,42,63,88,104,174,219,232,251,321,335
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 decrement(Counter storage counter) internal {
        uint256 value = counter._value;
        require(value > 0, "Counter: decrement overflow");
        unchecked {
            counter._value = value - 1;
        }
    }

function reset(Counter storage counter) internal {
        counter._value = 0;
    }

...
```

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.

L12 - Using Variables before Declaration

Criticality	Minor / Informative
Location	Feedcoin.sol#L179
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.

```
bytes32 r
```

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.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	Feedcoin.sol#L1058
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.

```
constructor(string memory name) EIP712(name, "1")
```

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.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	Feedcoin.sol#L184,239
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly {
                r := mload(add(signature, 0x20))
                s := mload(add(signature, 0x40))
                v := byte(0, mload(add(signature, 0x60)))
            }

assembly {
            s := and(vs,
0x7fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff)
            v := add(shr(255, vs), 27)
        }
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	Feedcoin.sol#L6,52,122,358,464,527,554,639,669,1027,1116,1159
Status	Unresolved

Description

If the contract includes multiple conflicting pragma directives, it may produce unexpected errors. To avoid this, it's important to include the correct pragma directive at the top of the contract and to ensure that it is the only pragma directive included in the contract.

```
pragma solidity ^0.8.0;

pragma solidity ^0.8.0;

pragma solidity ^0.8.2;
```

Recommendation

It is important to include only one pragma directive at the top of the contract and to ensure that it accurately reflects the version of Solidity that the contract is written in.

By including all required compiler options and flags in a single pragma directive, the potential conflicts could be avoided and ensure that the contract can be compiled correctly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	Feedcoin.sol#L6,52,122,358,464,527,554,639,669,1027,1116,1159
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 ^0.8.0;

pragma solidity ^0.8.0;

pragma solidity ^0.8.2;
```

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

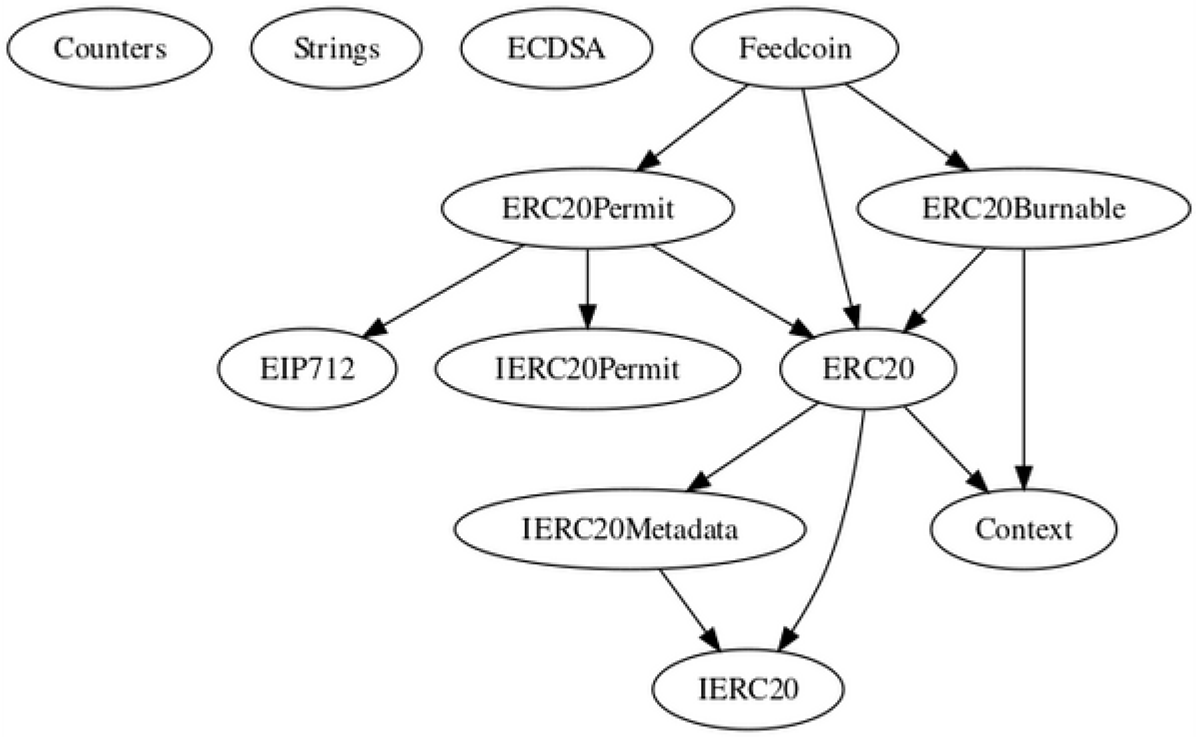
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Counters	Library			
	current	Internal		
	increment	Internal	✓	
	decrement	Internal	✓	
	reset	Internal	✓	
Strings	Library			
	toString	Internal		
	toHexString	Internal		
	toHexString	Internal		
ECDSA	Library			
	_throwError	Private		
	tryRecover	Internal		
	recover	Internal		
	tryRecover	Internal		
	recover	Internal		
	tryRecover	Internal		
	recover	Internal		
	toEthSignedMessageHash	Internal		
	toEthSignedMessageHash	Internal		
	toTypedDataHash	Internal		

EIP712	Implementation			
		Public	✓	-
	_domainSeparatorV4	Internal		
	_buildDomainSeparator	Private		
	_hashTypedDataV4	Internal		
IERC20Permit	Interface			
	permit	External	✓	-
	nonces	External		-
	DOMAIN_SEPARATOR	External		-
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		-

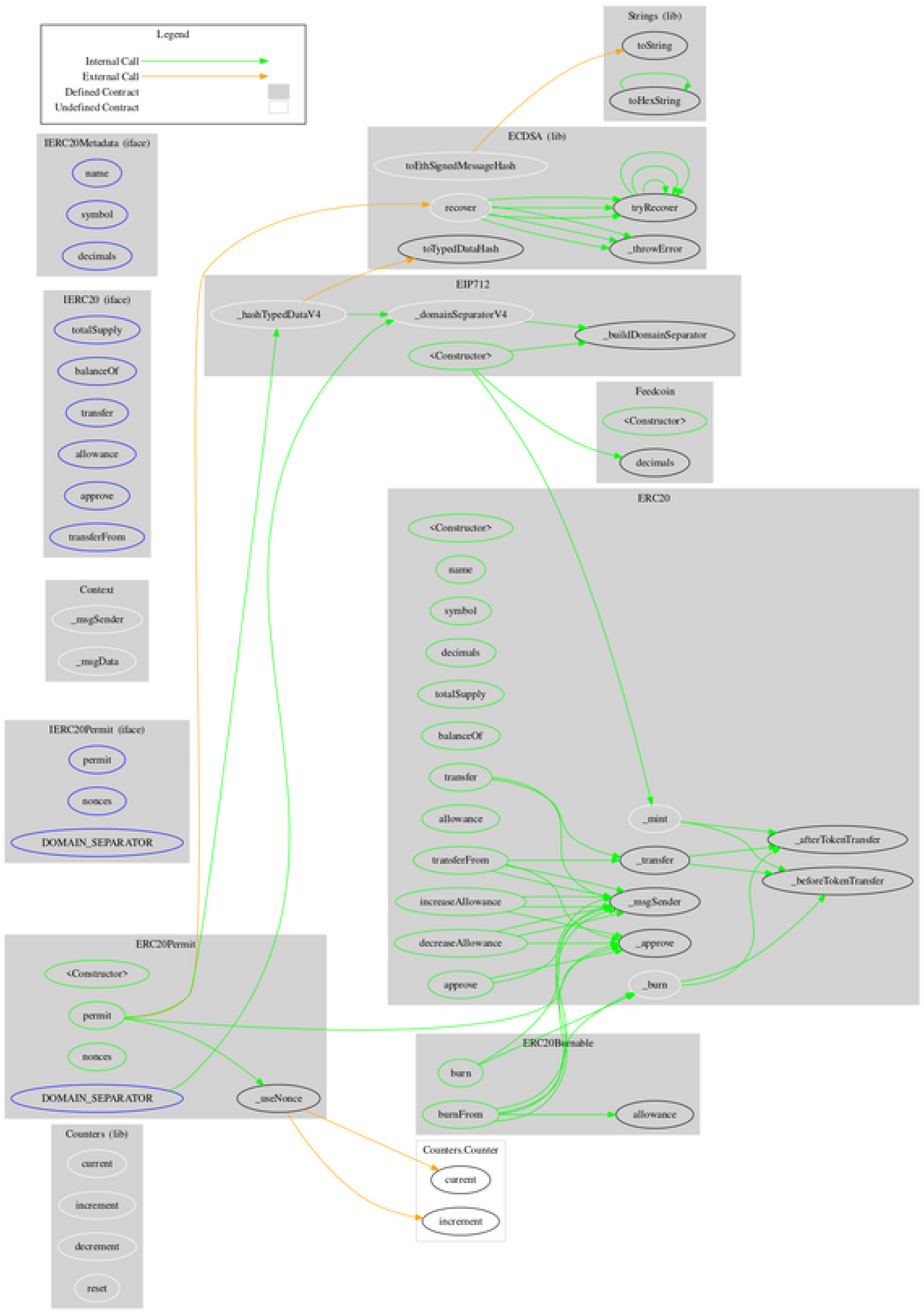
ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
ERC20Permit	Implementation	ERC20, IERC20Permit, EIP712		
		Public	✓	EIP712
	permit	Public	✓	-
	nonces	Public		-
	DOMAIN_SEPARATOR	External		-
	_useNonce	Internal	✓	

ERC20Burnable	Implementation	Context, ERC20		
	burn	Public	✓	-
	burnFrom	Public	✓	-
Feedcoin	Implementation	ERC20, ERC20Burnable, ERC20Permit		
		Public	✓	ERC20 ERC20Permit

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Strings (lib)
toString
toHexString
ECDSA (lib)
toEthSignedMessageHash
recover
tryRecover
toTypedDataHash
_throwError
IERC20Metadata (iface)
name
symbol
decimals
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
EIP712
_hashTypedDataV4
_domainSeparatorV4
_buildDomainSeparator
<Constructor>
Feedcoin
<Constructor>
decimals
Context
_msgSender
_msgData
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
transfer
allowance
transferFrom
increaseAllowance
decreaseAllowance
approve
_mint
_transfer
_msgSender
_approve
_burn
_afterTokenTransfer
_beforeTokenTransfer
IERC20Permit (iface)
permit
nonces
DOMAIN_SEPARATOR
ERC20Permit
<Constructor>
permit
nonces
DOMAIN_SEPARATOR
_useNonce
ERC20Burnable
burn
burnFrom
allowance
Counters (lib)
current
increment
decrement
reset
Counters.Counter
current
increment

Summary

Feed Coin 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

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