Cyberscope

Audit Report

BNDR Coin

February 2024

Network	BSC
Address	0x384b520ff25fec300cb612f0dddab92cb491c870
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
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved

Table of Contents

Review

Contract Name	BNDR
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	200 runs
Testing Deploy	https://testnet.bscscan.com/address/0xb5ce169cbe1bc2ee561a98579429649a032313c8
Explorer	https://bscscan.com/address/0x384b520ff25fec300cb612f0dddab92cb491c870
Address	0x384b520ff25fec300cb612f0dddab92cb491c870
Network	BSC
Symbol	BNDR
Decimals	18
Total Supply	210,000,000,000
Badge Eligibility	Yes

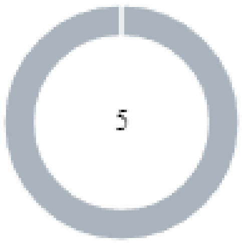
Audit Updates

Initial Audit	27 Feb 2024

Source Files

Filename	SHA256
contracts/BNDR.sol	ac44f4878bef52dc588f151d1da781b7b12080b88d2e83b1c693a5416826d5a4

Findings Breakdown



●	Critical	0
●	Medium	0
●	Minor / Informative	5

Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	0	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	5	0	0	0

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/BNDR.sol#L786,787,791,792,887,888
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.

```
mapping(address => bool isExcluded) public _excludedFromFee;
mapping(address => bool marketMaker) public _isMarketMaker;

uint256 public _burnFee = 10;
uint256 public _devFee = 30;

address[] memory _toExclude,

bool _value
```

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	contracts/BNDR.sol#L133,162,178,196,208,221,242,256,695
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 sendValue(address payable recipient, uint256 amount)
internal {
        if (address(this).balance < amount) {
            revert AddressInsufficientBalance(address(this));
        }

        (bool success, ) = recipient.call{value: amount}("");
        if (!success) {
            revert FailedInnerCall();
        }
    }

...
```

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	contracts/BNDR.sol#L806,807
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,
```

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/BNDR.sol#L378
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.

```
_pendingOwner = newOwner
```

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.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	contracts/BNDR.sol#L261
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 {
                let returndata_size := mload(returndata)
                revert(add(32, returndata), returndata_size)
            }
```

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.

Functions Analysis

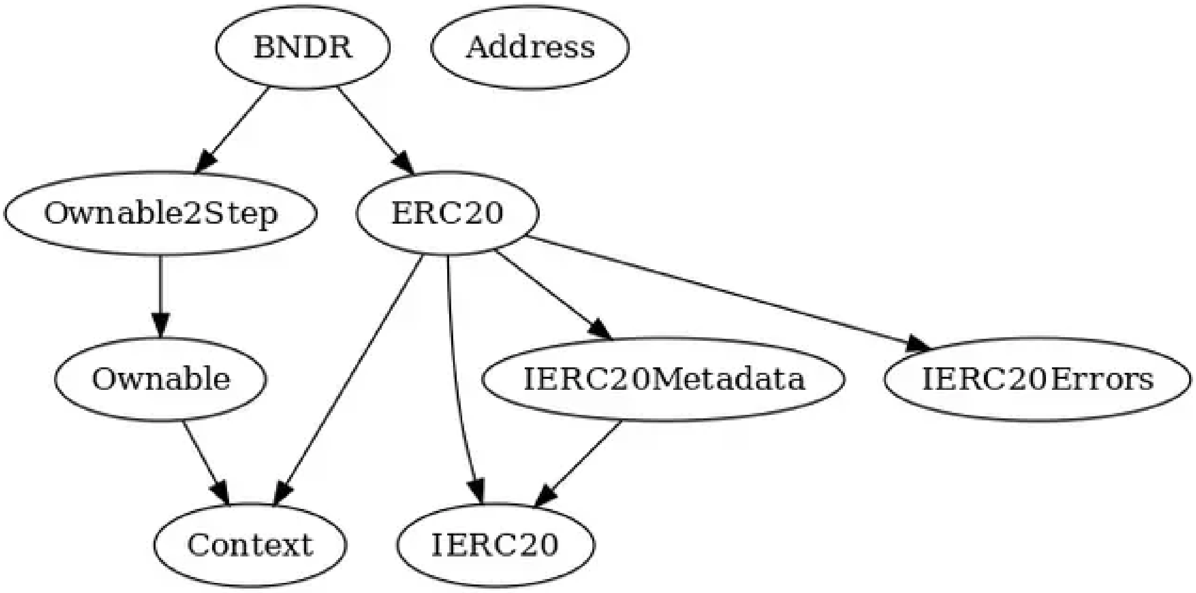
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Address	Library			
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	verifyCallResultFromTarget	Internal		

	verifyCallResult	Internal		
	_revert	Private		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
Ownable2Step	Implementation	Ownable		
	pendingOwner	Public		-
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
	acceptOwnership	Public	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
IERC20Errors	Interface			

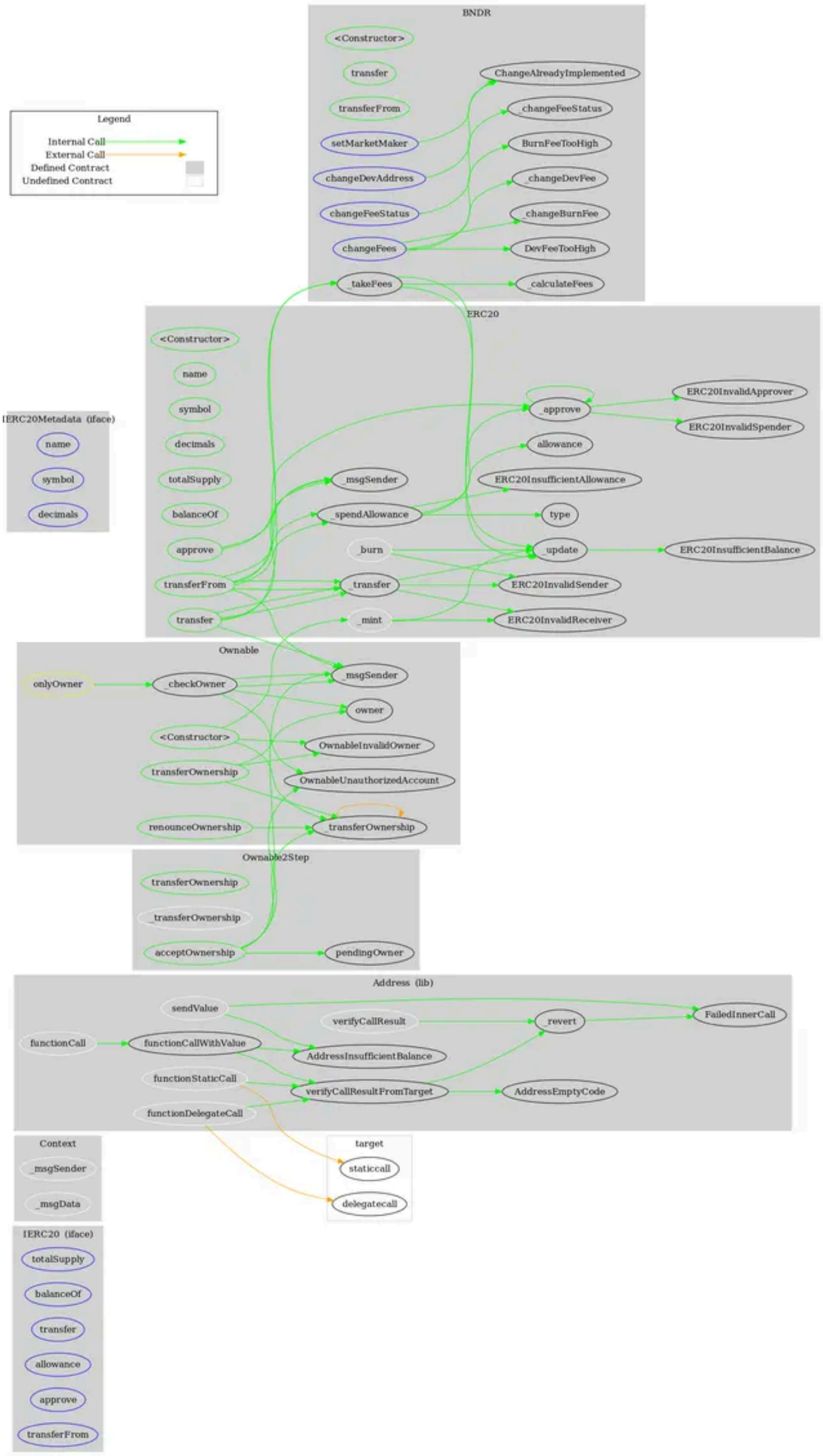
ERC20	Implementation	Context, IERC20, IERC20Metadata, IERC20Errors		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	_transfer	Internal	✓	
	_update	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_approve	Internal	✓	
	_spendAllowance	Internal	✓	
BNDR	Implementation	ERC20, Ownable2Step		

		Public	✓	ERC20 Ownable
	transfer	Public	✓	-
	transferFrom	Public	✓	-
	_takeFees	Internal	✓	
	_calculateFees	Internal		
	setMarketMaker	External	✓	onlyOwner
	changeFeeStatus	External	✓	onlyOwner
	_changeFeeStatus	Internal	✓	
	changeDevAddress	External	✓	onlyOwner
	changeFees	External	✓	onlyOwner
	_changeBurnFee	Internal	✓	
	_changeDevFee	Internal	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
BNDR
<Constructor>
transfer
transferFrom
setMarketMaker
changeDevAddress
changeFeeStatus
changeFees
_takeFees
ChangeAlreadyImplemented
_changeFeeStatus
BurnFeeTooHigh
_changeDevFee
_changeBurnFee
DevFeeTooHigh
_calculateFees
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
approve
transferFrom
transfer
_msgSender
_spendAllowance
_burn
_transfer
_mint
_approve
allowance
ERC20InsufficientAllowance
type
_update
ERC20InvalidSender
ERC20InvalidReceiver
ERC20InvalidApprover
ERC20InvalidSpender
ERC20InsufficientBalance
IERC20Metadata (iface)
name
symbol
decimals
Ownable
onlyOwner
_checkOwner
<Constructor>
transferOwnership
renounceOwnership
_msgSender
owner
OwnableInvalidOwner
OwnableUnauthorizedAccount
_transferOwnership
Ownable2Step
transferOwnership
_transferOwnership
acceptOwnership
pendingOwner
Address (lib)
sendValue
functionCall
functionCallWithValue
functionStaticCall
functionDelegateCall
verifyCallResult
AddressInsufficientBalance
verifyCallResultFromTarget
_revert
FailedInnerCall
AddressEmptyCode
Context
_msgSender
_msgData
target
staticcall
delegatecall
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom

Summary

BNDR Coin contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. BNDR 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

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