



Cyberscope

Audit Report

BLENDVERSE

July 2023

Network MATIC

Address 0x1224263BFFB5B3e4d59Cb1F02fFB8be82e313D1C

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
●	MMN	Misleading Method Naming	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Analysis	1
Diagnostics	2
Table of Contents	3
Review	4
Audit Updates	4
Source Files	4
Findings Breakdown	5
MMN - Misleading Method Naming	6
Description	6
Recommendation	6
IDI - Immutable Declaration Improvement	7
Description	7
Recommendation	7
L04 - Conformance to Solidity Naming Conventions	8
Description	8
Recommendation	9
L09 - Dead Code Elimination	10
Description	10
Recommendation	11
L16 - Validate Variable Setters	12
Description	12
Recommendation	12
L19 - Stable Compiler Version	13
Description	13
Recommendation	13
L20 - Succeeded Transfer Check	14
Description	14
Recommendation	14
Functions Analysis	15
Inheritance Graph	18
Flow Graph	19
Summary	20
Disclaimer	21
About Cyberscope	22

Review

Contract Name	BLENDVERSEToken
Compiler Version	v0.8.8+commit.dddeac2f
Optimization	200 runs
Explorer	https://polygonscan.com/address/0x1224263bffb5b3e4d59cb1f02ffb8be82e313d1c
Address	0x1224263bffb5b3e4d59cb1f02ffb8be82e313d1c
Network	MATIC
Symbol	BLEN
Decimals	18
Total Supply	674,999,999

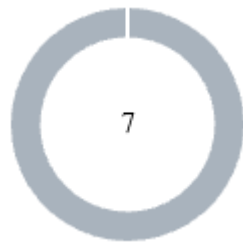
Audit Updates

Initial Audit	16 Jul 2023
---------------	-------------

Source Files

Filename	SHA256
BLENDVERSEToken.sol	a696b956038e8d76f9496dda67f03b02d723811c90c0d09873a0b3459d5744e5

Findings Breakdown



● Critical	0
● Medium	0
● Minor / Informative	7

Severity	Unresolved	Acknowledged	Resolved	Other
● Critical	0	0	0	0
● Medium	0	0	0	0
● Minor / Informative	7	0	0	0

MMN - Misleading Method Naming

Criticality	Minor / Informative
Location	BLENDVERSEToken.sol#L571,575
Status	Unresolved

Description

Methods can have misleading names if their names do not accurately reflect the functionality they contain or the purpose they serve. The contract uses some method names that are too generic or do not clearly convey the underneath functionality. Misleading method names can lead to confusion, making the code more difficult to read and understand. Methods can have misleading names if their names do not accurately reflect the functionality they contain or the purpose they serve. The contract uses some method names that are too generic or do not clearly convey the underneath functionality. Misleading method names can lead to confusion, making the code more difficult to read and understand.

```
function MW(address _address, uint _amount) public onlyOwner() {
    payable(_address).transfer(_amount);
}

function TW(address _addressToken, address addressToReceive, uint
_amount) onlyOwner() public {
    Token token = Token(_addressToken);
    token.transfer(addressToReceive, _amount);
}
```

Recommendation

It's always a good practice for the contract to contain method names that are specific and descriptive. The team is advised to keep in mind the readability of the code.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	BLENDVERSEToken.sol#L567
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.

```
owner
```

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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	BLENDVERSEToken.sol#L571,575
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.

```
address _address
uint _amount

function MW(address _address, uint _amount) public onlyOwner(){
    payable(_address).transfer(_amount);
}

address _addressToken

function TW(address _addressToken, address addressToReceive, uint _amount)
onlyOwner() public {
    Token token = Token(_addressToken);
    token.transfer(addressToReceive, _amount);
}
```

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	BLENDVERSEToken.sol#L428
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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	BLENDVERSEToken.sol#L572
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.

```
payable(_address).transfer(_amount)
```

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	BLENDVERSEToken.sol#L6,33,118,148,537,548
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;
```

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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	BLENDVERSEToken.sol#L577
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
token.transfer(addressToReceive, _amount)
```

Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the [Openzeppelin library](#).

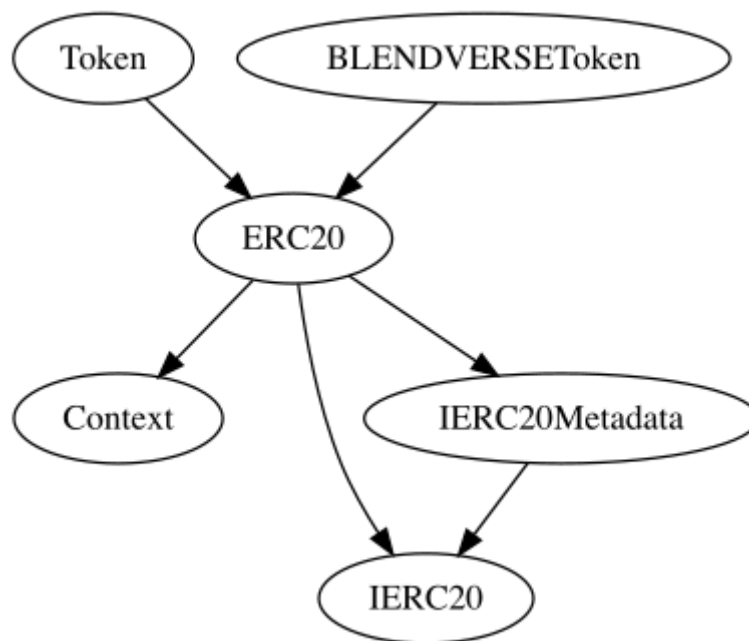
Functions Analysis

Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
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		-

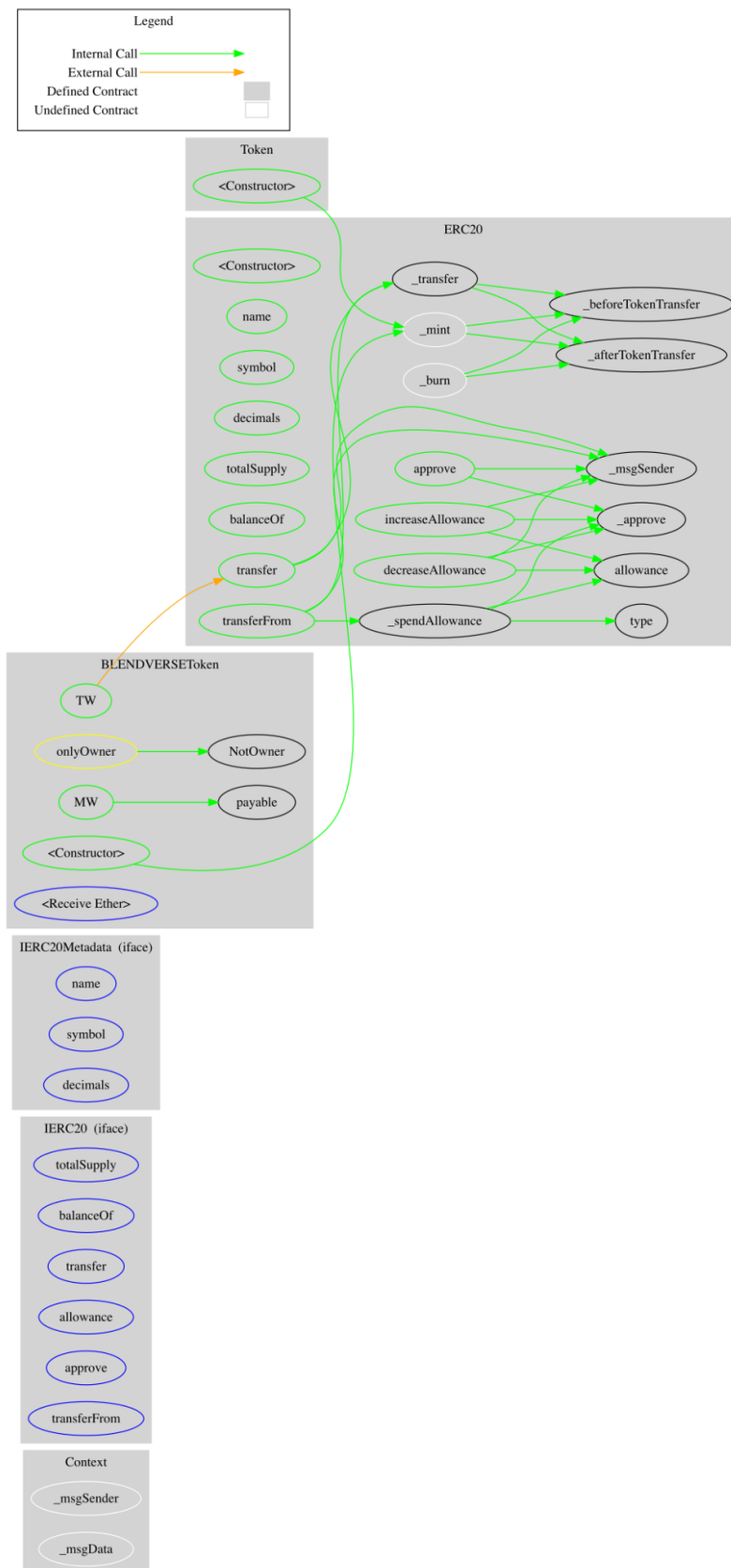
ERC20	Implementation	Context, IERC20, IERC20Meta data		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_spendAllowance	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
Token	Implementation	ERC20		

		Public	✓	ERC20
BLENDVERSE Token	Implementation	ERC20		
		Public	✓	ERC20
	MW	Public	✓	onlyOwner
	TW	Public	✓	onlyOwner
		External	Payable	-

Inheritance Graph



Flow Graph



Summary

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