Cyberscope

Audit Report

Borg Original

December 2023

Network BSC

Address 0x53dff783f3ddda0c50ebfed8fea6cbd9f776abd1

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
	RED	Redundant Event Declaration	Unresolved
	MEE	Missing Events Emission	Unresolved
	RSW	Redundant Storage Writes	Unresolved
	RCS	Redundant Conditional Statement	Unresolved
	RFW	Redundant Fee Wallets	Unresolved
	MAU	Misleading Address Usage	Unresolved
	RMTF	Redundant Max Transaction Functionality	Unresolved
	PAV	Pair Address Validation	Unresolved
	L02	State Variables could be Declared Constant	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L05	Unused State Variable	Unresolved
	L09	Dead Code Elimination	Unresolved
	L13	Divide before Multiply Operation	Unresolved

Table of Contents

Review

Contract Name	BorgOriginal
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	200 runs
Explorer	https://bscscan.com/address/0x53dfF783f3DDDa0c50eBfed8Fea6cBd9f776ABD1
Address	0x53dff783f3ddda0c50ebfed8fea6cbd9f776abd1
Network	BSC
Symbol	BORG
Decimals	18
Total Supply	70,000,000,000,000

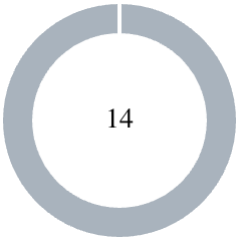
Audit Updates

Initial Audit	25 Oct 2023 https://github.com/cyberscope-io/audits/blob/main/borg/v1/audit.pdf
Corrected Phase 2	02 Nov 2023 https://github.com/cyberscope-io/audits/blob/main/borg/v2/audit.pdf
Corrected Phase 3	06 Nov 2023 https://github.com/cyberscope-io/audits/blob/main/borg/v3/audit.pdf
Corrected Phase 4	14 Nov 2023 https://github.com/cyberscope-io/audits/blob/main/borg/v4/audit.pdf
Corrected Phase 5	17 Nov 2023 https://github.com/cyberscope-io/audits/blob/main/borg/v5/audit.pdf
Corrected Phase 6	13 Dec 2023

Source Files

Filename	SHA256
BorgOriginal.sol	80d970747289aed192c201e0f8a0c02ef2a45e52c02006028af91d7cc60aaa68

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	14

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	14	0	0	0

RED - Redundant Event Declaration

Criticality	Minor / Informative
Location	contracts/BorgOriginal.sol#L694
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 declares certain events in its code. However, these events are not emitted within the contract's functions. As a result, these declared events are redundant and serve no purpose within the contract's current implementation.

```
event Payout(
    address indexed marketingWallet,
    uint8 marketingAmount,
    address indexed treasuryWallet,
    uint8 treasuryAmount
);
```

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.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	contracts/BorgOriginal.sol#L731,855,860,862,865
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.

```
isExchangeAddress[_address] = _status;
reflectionAddress1 = _address;
liquidityWallet = _address;
burnWallet = _address;
reflectionAddress2 = _address;
```

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.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	contracts/BorgOriginal.sol#L731
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.

```
isExchangeAddress[_address] = _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.

RCS - Redundant Conditional Statement

Criticality	Minor / Informative
Location	contracts/BorgOriginal.sol#L805
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 includes a condition to check if the recipient is not excluded from fees in the context of a buy transaction. However, this check is redundant, as the condition `isExchangeAddress[sender]` along with the `!_isExcludedFromFee[sender] && !_isExcludedFromFee[recipient]` condition can handle the fee calculations appropriately. In other words, if the recipient is excluded, none of the if-else blocks related to fee calculation will be executed, resulting in a zero fee. On the other hand, if the recipient is not excluded, the buy transaction block will set the fee amount to zero. As a result, the `!_isExcludedFromFee[recipient]` check is redundant.

```
uint256 feeAmount = 0;
// Check if it's a buy transaction (sender is an exchange and
recipient is not excluded from fee)
if (isExchangeAddress[sender] && !_isExcludedFromFee[recipient]) {
    // Apply zero buy tax
    feeAmount = 0;
} else if (
    !_isExcludedFromFee[sender] && !_isExcludedFromFee[recipient]
) {
    // For normal transfers and sell transactions, calculate the
fee
    feeAmount = (amount * feeRate) / FEE_DENOMINATOR;
    _balances[address(this)] += feeAmount;
    emit Transfer(sender, address(this), feeAmount);
}
```

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.

RFW - Redundant Fee Wallets

Criticality	Minor / Informative
Location	contracts/BorgOriginal.sol#L663,664,665,666,667,668,669,681,682,683,684,685,686
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 declares specific wallet addresses along with their corresponding fee percentages, such as `marketingWallet`, `treasuryWallet`, `reflectionAddress1`, `reflectionAddress2`, `liquidityWallet`, and `burnWallet`. However, the fees collected by the contract are not utilized or transferred to these declared wallets, making their declaration redundant.

```
address public marketingWallet =
0x8534C63B9856Fe2Ed93aDf507600332B244A2fdC;
address public treasuryWallet =
0x8136100798fdE72de63166a328eaE6aBE7368612;
address public reflectionAddress1 = address(0);
address public reflectionAddress2 =
    0xA02B448630c5C0bd476BB5e455948F2A56A5828A;
address public liquidityWallet = address(0);
address public burnWallet = address(0);

uint256 reflectionFee1 = 4;
uint256 reflectionFee2 = 4;
uint256 marketingFee = 4;
uint256 treasuryFee = 4;
uint256 liquidityFee = 3;
uint256 burnFee = 1;
```

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.

MAU - Misleading Address Usage

Criticality	Minor / Informative
Location	contracts/BorgOriginal.sol#L863
Status	Unresolved

Description

The contract contains a variable called `burnWallet` address to represent a specific type of address, commonly acknowledged in the blockchain for a particular purpose. However, this wallet address within this contract is mutable, meaning it can be altered. As a result, the designated address may not consistently serve its conventional purpose, potentially leading to unintended behaviors within the contract's operation. This mutable design diverges from the standard practice of utilizing a fixed, immutable address for such purposes, thereby introducing a layer of complexity and potential risk in the contract's functionality.

```
function setBurnWallet(address _address) public onlyOwner {
    require(_address != address(0), "Cannot be zero address");
    burnWallet = _address;
}
```

Recommendation

It's always a good practice for the contract to contain variable names that are specific and descriptive. The team is advised to keep in mind the clarity and comprehensibility of the code to ensure that it accurately reflects the intended functionality. The designated address, which reflects a specific purpose within the contract, should ideally be immutable to maintain consistency in its functionality and to adhere to common practices, thereby reducing the potential for unexpected behaviors or vulnerabilities within the contract's operation.

RMTF - Redundant Max Transaction Functionality

Criticality	Minor / Informative
Location	contracts/BorgOriginal.sol#L677,837
Status	Unresolved

Description

The contract introduces the mapping variable `_isExcludedFromMaxTransactionAmount` and the function `excludeFromMaxTransactionAmount`. However, there is no implementation or constraint related to the maximum transaction amount within the contract's transfer function. Consequently, the declaration of this variable and the associated function is redundant.

```
mapping(address => bool) _isExcludedFromMaxTransactionAmount;

function excludeFromMaxTransactionAmount(
    address account,
    bool isEx
) external onlyOwner {
    require(
        _isExcludedFromMaxTransactionAmount[account] != isEx,
        "Account is already the desired state"
    );
    _isExcludedFromMaxTransactionAmount[account] = isEx;
    emit ExcludeFromMaxTransactionAmount(account, isEx);
}
```

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.

PAV - Pair Address Validation

Criticality	Minor / Informative
Location	contracts/BorgOriginal.sol#L726
Status	Unresolved

Description

The `setExchangeAddress` function allows the contract owner to set any arbitrary value without validation to the `isExchangeAddress` mapping, which is supposed to hold Uniswap pair addresses. This lack of validation can lead to unintended behavior, including the potential disruption of the contract's intended functionality.

```
function setExchangeAddress(
    address _address,
    bool _status
) public onlyOwner {
    require(_address != address(0), "Cannot be zero address");
    isExchangeAddress[_address] = _status;
}
```

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.

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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	BorgOriginal.sol#L660,663,664,680,681,682,683,684,685
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.

```
uint8 public feeRate = 20
address public marketingWallet =
0x8534C63B9856Fe2Ed93aDf507600332B244A2fdC
address public treasuryWallet =
0x8136100798fdE72de63166a328eaE6aBE7368612
uint256 reflectionFee1 = 4
uint256 reflectionFee2 = 4
uint256 marketingFee = 4
uint256 treasuryFee = 4
uint256 liquidityFee = 3
uint256 burnFee = 1
```

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	BorgOriginal.sol#L473,671,672,676,726,727,852,857,862,867
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 DOMAIN_SEPARATOR() external view returns (bytes32);
mapping(address => uint256) _balances
mapping(address => mapping(address => uint256)) _allowances
mapping(address => bool) _isExcludedFromMaxTransactionAmount
address _address
bool _status
```

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	BorgOriginal.sol#L677
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.

```
mapping(address => bool) automatedMarketMakerPairs
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	BorgOriginal.sol#L41,70,86,104,116,129,150,164,508,516,532,545,568,593,612,761
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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	BorgOriginal.sol#L768,769,771,773,775,776,778
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
uint256 feeAmount = (amount * feeRate) / FEE_DENOMINATOR
uint256 liquidityFeeShare = (feeAmount * liquidityFee) /
            FEE_DENOMINATOR
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	BorgOriginal.sol#L169
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

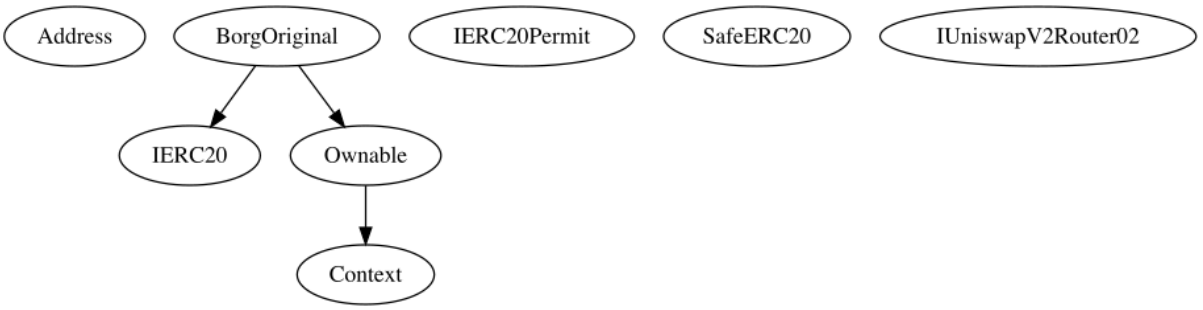
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Address	Library			
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	verifyCallResultFromTarget	Internal		
	verifyCallResult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

	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
IERC20Permit	Interface			
	permit	External	✓	-
	nonces	External		-
	DOMAIN_SEPARATOR	External		-
SafeERC20	Library			
	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeIncreaseAllowance	Internal	✓	
	safeDecreaseAllowance	Internal	✓	
	forceApprove	Internal	✓	
	_callOptionalReturn	Private	✓	

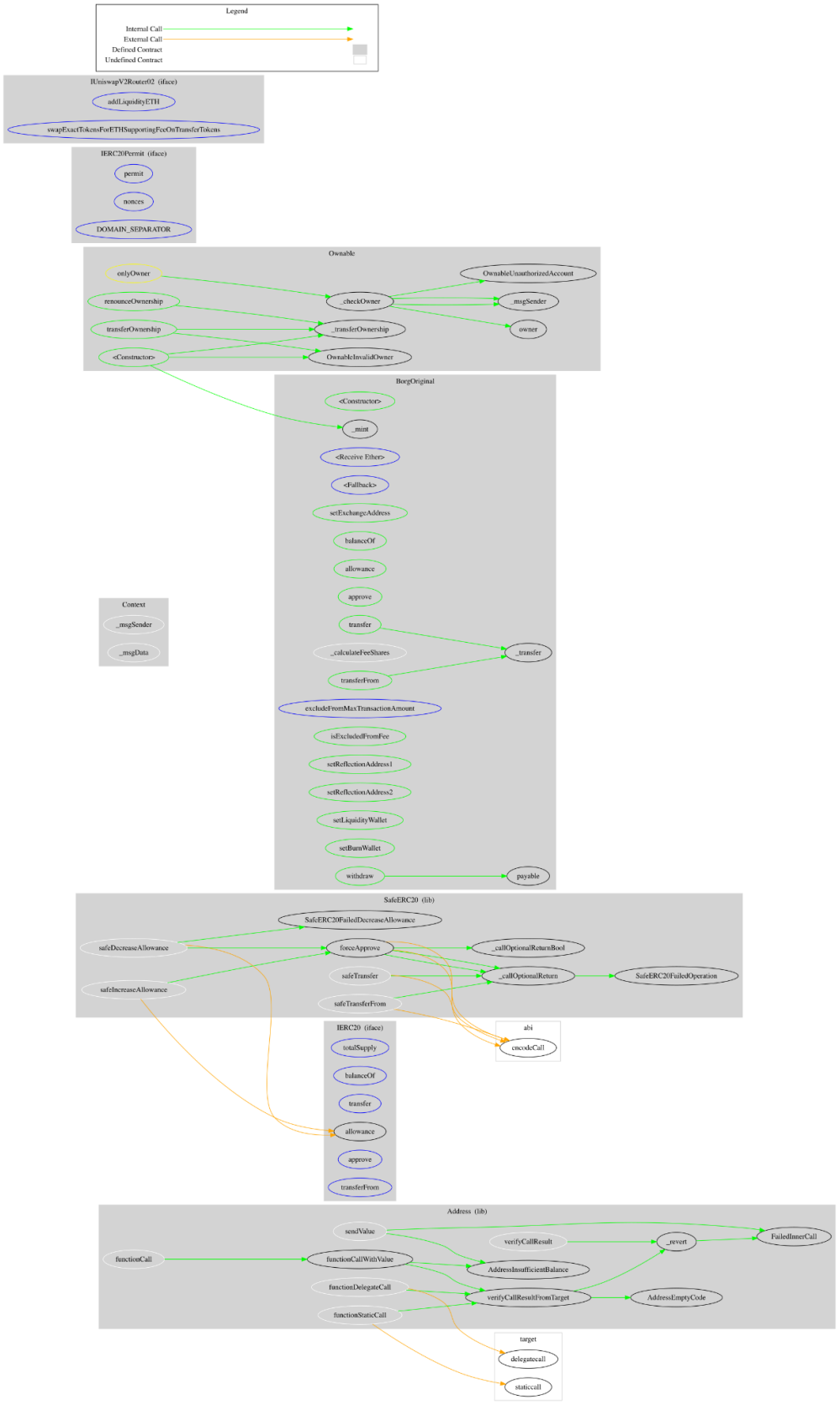
	_callOptionalReturnBool	Private	✓	
IUniswapV2Router02	Interface			
	addLiquidityETH	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
BorgOriginal	Implementation	IERC20, Ownable		
		Public	✓	Ownable
	_mint	Internal	✓	
		External	Payable	-
		External	✓	-
	setExchangeAddress	Public	✓	onlyOwner
	balanceOf	Public		-
	allowance	Public		-
	approve	Public	✓	-
	transfer	Public	✓	-
	_calculateFeeShares	Internal		
	_transfer	Internal	✓	
	transferFrom	Public	✓	-
	excludeFromMaxTransactionAmount	External	✓	onlyOwner
	isExcludedFromFee	Public		-
	setReflectionAddress1	Public	✓	onlyOwner
	setReflectionAddress2	Public	✓	onlyOwner

	setLiquidityWallet	Public	✓	onlyOwner
	setBurnWallet	Public	✓	onlyOwner
	withdraw	Public	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IUniswapV2Router02 (iface)
addLiquidityETH
swapExactTokensForETHSupportingFeeOnTransferTokens
IERC20Permit (iface)
permit
nonces
DOMAIN_SEPARATOR
Ownable
onlyOwner
renounceOwnership
transferOwnership
<Constructor>
OwnableUnauthorizedAccount
_checkOwner
_msgSender
owner
_transferOwnership
OwnableInvalidOwner
BorgOriginal
<Constructor>
_mint
<Receive Ether>
<Fallback>
setExchangeAddress
balanceOf
allowance
approve
transfer
_calculateFeeShares
_transfer
transferFrom
excludeFromMaxTransactionAmount
isExcludedFromFee
setReflectionAddress1
setReflectionAddress2
setLiquidityWallet
setBurnWallet
withdraw
payable
Context
_msgSender
_msgData
SafeERC20 (lib)
SafeERC20FailedDecreaseAllowance
safeDecreaseAllowance
forceApprove
_callOptionalReturnBool
safeTransfer
_callOptionalReturn
SafeERC20FailedOperation
safeIncreaseAllowance
safeTransferFrom
abi
encodeCall
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Address (lib)
sendValue
verifyCallResult
_revert
FailedInnerCall
functionCall
functionCallWithValue
AddressInsufficientBalance
functionDelegateCall
verifyCallResultFromTarget
AddressEmptyCode
functionStaticCall
target
delegatecall
staticcall

Summary

Borg Original contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. Borg Original is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler errors 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. There is also a limit of max 20% transfer and sell fees.

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