Cyberscope

Audit Report

Borg Original

October 2023

SHA256 0a14af2f5d4b735d086235e0dc1ad644572096bcc62faf8f6fd2985e36a7c08c

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	Unresolved
●	MT	Mints Tokens	Unresolved
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	TSD	Total Supply Diversion	Unresolved
●	RRI	Redundant ReentrancyGuard Import	Unresolved
●	UFD	Unallocated Fee Discrepancy	Unresolved
●	MTI	Misleading Tax Implementation	Unresolved
●	DDP	Decimal Division Precision	Unresolved
●	PMI	Potential Missing Implementation	Unresolved
●	EIS	Excessively Integer Size	Unresolved
●	MCM	Misleading Comment Messages	Unresolved
●	RED	Redudant Event Declaration	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L22	Potential Locked Ether	Unresolved

Table of Contents

Review

Testing Deploy	https://testnet.bscscan.com/address/0x0654bc2a7a894e4c75976e50c4782dd3cbea264d

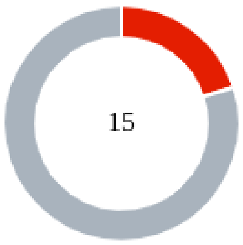
Audit Updates

Initial Audit	25 Oct 2023

Source Files

Filename	SHA256
BorgOriginal.sol	0a14af2f5d4b735d086235e0dc1ad644572096bcc62faf8f6fd2985e36a7c08c

Findings Breakdown



●	Critical	3
●	Medium	0
●	Minor / Informative	12

Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	3	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	12	0	0	0

ELFM - Exceeds Fees Limit

Criticality	Critical
Location	BorgOriginal.sol#L419
Status	Unresolved

Description

The contract is designed to calculate the `netAmount` variable, which represents the tax amount based on either the `sellTax` or the `totalFee` applied. However, instead of sending to the `recipient` the `amount` transferred minus the `netAmount` (which represents the fee), the contract is setting the balance of the `recipient` by adding the `netAmount`. As a result, if the tax fee is set up to 20%, the recipient receives only up to 20% of the transferred amount. This leads to an effective fee of 80%, which exceeds increase over the allowed limit of 25%.

```
   uint256 totalFee = 0;
    uint256 netAmount = 0;

    if (recipient != address(0)) {  // If it's a sell, apply
20% tax
        netAmount = (amount * sellTax) / 100;
        totalFee = sellTax;
    } else {
        totalFee = reflectionFee1 + reflectionFee2 +
marketingFee + treasuryFee + liquidityFee + burnFee;
        netAmount = (amount * totalFee) / 100;
    }

    // Deduct the fee and send the net amount
    _balances[sender] = _balances[sender] - amount;
    _balances[recipient] = _balances[recipient] + netAmount;
```

Recommendation

The contract could embody a check for the maximum acceptable value. It is recommended to modify the contract's logic such that, instead of adding the `netAmount` to the `_balances` of the `recipient`, the contract should add `amount - netAmount`.

This results to the total transferred amount minus the fees, ensuring that the recipient receives the correct amount after tax deductions.

MT - Mints Tokens

Criticality	Critical
Location	BorgOriginal.sol#L353
Status	Unresolved

Description

The contract owner has the authority to mint tokens. The owner may take advantage of it by calling the `mint` function. As a result, the contract tokens will be highly inflated.

```
    function mint(address account, uint256 amount) external
onlyOwner {
        _mint(account, amount);
    }
```

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.

TSD - Total Supply Diversion

Criticality	Critical
Location	BorgOriginal.sol#L345
Status	Unresolved

Description

The total supply of a token is the total number of tokens that have been created, while the balances of individual accounts represent the number of tokens that an account owns. The total supply and the balances of individual accounts are two separate concepts that are managed by different variables in a smart contract. These two entities should be equal to each other.

In the contract, the amount that is added to the total supply does not equal the amount that is added to the balances. As a result, the sum of balances is diverse from the total supply.

Specificcaly the contract is initializing the `totalSupply` variable with a value of `7000000000000000000000000000000`. Subsequently, the contract also mints an additional amount of tokens of the same value, effectively doubling the intended supply. This behavior can lead to unintended inflation of the token supply and potential misunderstandings about the contract's actual total supply.

```
uint256 public totalSupply = 7000000000000000000000000000000;
..
_mint(msg.sender, 7000000000000000000000000000000); // 5 trillion
tokens as initial supply
}

function _mint(address account, uint256 amount) internal {
    require(account != address(0), "Mint to the zero address");
    totalSupply += amount;
    _balances[account] += amount;
    emit Transfer(address(0), account, amount);
}
```

Recommendation

The total supply and the balance variables are separate and independent from each other. The total supply represents the total number of tokens that have been created, while the balance mapping stores the number of tokens that each account owns. The sum of balances should always equal the total supply. It is recommended to initialize the `_totalSupply` variable to `zero` and increase it only once the minting operation occurs. This ensures clarity in the contract's token supply and prevents unintended duplication of minted tokens.

RRI - Redundant ReentrancyGuard Import

Criticality	Minor / Informative
Location	BorgOriginal.sol#L1
Status	Unresolved

Description

The contract is using the `ReentrancyGuard` library. However, this library is not being utilized in any of the contract's functions or modifiers. Therefore the usage of this library is redundant and can lead to unnecessary gas increase.

```
abstract contract ReentrancyGuard {
...
}
```

Recommendation

It is recommended to remove the ReentrancyGuard library to streamline the contract's code and eliminate unnecessary dependencies.

UFD - Unallocated Fee Discrepancy

Criticality	Minor / Informative
Location	BorgOriginal.sol#L413
Status	Unresolved

Description

The contract is designed to calculate various fees, including `liquidityFee` and `burnFee`, which are then added to the `totalFee` amount. Subsequently, this total fee is deducted from the `amount` intended for the `recipient`. However, while the fees `reflectionFee1`, `reflectionFee2`, `marketingFee`, and `treasuryFee` are allocated to specific addresses, the `liquidityFee` and `burnFee` are not added to any account. This leads to a discrepancy where these fees are deducted but not effectively allocated or utilized within the contract.

```
       totalFee = reflectionFee1 + reflectionFee2 + marketingFee +
treasuryFee + liquidityFee + burnFee;
        netAmount = (amount * totalFee) / 100;
    }

    // Deduct the fee and send the net amount
     _balances[sender] = _balances[sender] - amount;
     _balances[recipient] = _balances[recipient] + netAmount;
     _balances[reflectionAddress1] = _balances[reflectionAddress1] +
(amount * reflectionFee1) / 100;
     _balances[reflectionAddress2] = _balances[reflectionAddress2] +
(amount * reflectionFee2) / 100;
     _balances[marketingWallet] = _balances[marketingWallet] + (amount *
marketingFee) / 100;
     _balances[treasuryWallet] = _balances[treasuryWallet] + (amount *
treasuryFee) / 100;
```

Recommendation

It is recommended to ensure that all fees deducted from the transferred amount are handled correctly. Specifically, the contract should either allocate the `liquidityFee` and `burnFee` to their respective destinations or include code functioanlity to handle these unallocated fees.

MTI - Misleading Tax Implementation

Criticality	Minor / Informative
Location	BorgOriginal.sol#L409
Status	Unresolved

Description

The contract is designed with an intention to apply a tax if a transfer is a sale, as indicated by the comment. However, the actual implementation of the if statement does not align with the comment's documentation. Instead of applying the tax exclusively to sell transactions, the contract applies the tax to every transfer where the `recipient` is not the zero address. This discrepancy between the comment and the actual code can lead to confusion and unintended financial consequences for users interacting with the contract.

```
    if (recipient != address(0)) {  // If it's a sell, apply 20% 
tax
        netAmount = (amount * sellTax) / 100;
        totalFee = sellTax;
    } else {
        totalFee = reflectionFee1 + reflectionFee2 + 
marketingFee + treasuryFee + liquidityFee + burnFee;
        netAmount = (amount * totalFee) / 100;
    }
```

Recommendation

It is recommended to reconsider the code implementation. If the intended function is to apply fees only to sale transactions, then the contract should check if the recipient is a specific sale address or another condition that accurately represents a sale, rather than checking if the recipient is not the zero address.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	BorgOriginal.sol#L420
Status	Unresolved

Description

Division of decimal (fixed point) numbers can result in rounding errors due to the way that division is implemented in Solidity. Thus, it may produce issues with precise calculations with decimal numbers.

Solidity represents decimal numbers as integers, with the decimal point implied by the number of decimal places specified in the type (e.g. decimal with 18 decimal places). When a division is performed with decimal numbers, the result is also represented as an integer, with the decimal point implied by the number of decimal places in the type. This can lead to rounding errors, as the result may not be able to be accurately represented as an integer with the specified number of decimal places.

Hence, the splitted shares will not have the exact precision and some funds may not be calculated as expected.

```
_balances[reflectionAddress1] = _balances[reflectionAddress1] +
(amount * reflectionFee1) / 100;
_balances[reflectionAddress2] = _balances[reflectionAddress2] +
(amount * reflectionFee2) / 100;
_balances[marketingWallet] = _balances[marketingWallet] +
(amount * marketingFee) / 100;
_balances[treasuryWallet] = _balances[treasuryWallet] + (amount
* treasuryFee) / 100;
```

Recommendation

The team is advised to take into consideration the rounding results that are produced from the solidity calculations. The contract could calculate the subtraction of the divided funds in the last calculation in order to avoid the division rounding issue.

PMI - Potential Missing Implementation

Criticality	Minor / Informative
Location	BorgOriginal.sol#L424
Status	Unresolved

Description

The contract contains a comment indicating an intention to "Implement logic for liquidity and burn fees." However, there is no actual implementation that corresponds to this comment. This discrepancy can lead to confusion and potential misunderstandings about the contract's intended functionality and its actual behavior.

```
// Implement logic for liquidity and burn fees
```

Recommendation

It is recommended to either remove the comment line since there is no actual implementation to be described or to implement the corresponding functionality to ensure clarity and consistency within the contract's source code.

EIS - Excessively Integer Size

Criticality	Minor / Informative
Location	BorgOriginal.sol#L321
Status	Unresolved

Description

The contract is using a bigger unsigned integer data type that the maximum size that is required. By using an unsigned integer data type larger than necessary, the smart contract consumes more storage space and requires additional computational resources for calculations and operations involving these variables. This can result in higher transaction costs, longer execution times, and potential scalability bottlenecks.

```
   uint256 public reflectionFee1 = 4; // 4% Reflection fee to
first mechanism
    uint256 public reflectionFee2 = 4; // 4% Reflection fee to
second mechanism
    uint256 public marketingFee = 4; // 4% Marketing fee
    uint256 public treasuryFee = 4; // 4% Treasury fee
    uint256 public liquidityFee = 3; // 3% Liquidity fee
    uint256 public burnFee = 1; // 1% Burn fee
    uint256 public sellTax = 20; // 20% Sell tax
```

Recommendation

To address the inefficiency associated with using an oversized unsigned integer data type, it is recommended to accurately determine the required size based on the range of values the variable needs to represent.

MCM - Misleading Comment Messages

Criticality	Minor / Informative
Location	BorgOriginal.sol#L341
Status	Unresolved

Description

The contract is using misleading comment messages. These comment messages do not accurately reflect the actual implementation, making it difficult to understand the source code. As a result, the users will not comprehend the source code's actual implementation.

Specifically, the comment suggests an initial supply of 5 trillion tokens, but the code mints a total of 7 trillion tokens. Such discrepancies between comments and actual implementation can lead to confusion and misunderstandings about the contract's behavior.

```
_mint(msg.sender, 70000000000000000000000000000000); // 5
trillion tokens as initial supply
```

Recommendation

The team is advised to carefully review the comment in order to reflect the actual implementation. To improve code readability, the team should use more specific and descriptive comment messages.

RED - Redudant Event Declaration

Criticality	Minor / Informative
Location	BorgOriginal.sol#L337
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 events `GameRewardClaimed` and `TokensSpent` are declared and not being used in the contract. As a result, are redundant.

```
event GameRewardClaimed(address indexed user, uint256 amount);
event TokensSpent(address indexed user, address gameAddress,
uint256 amount);
```

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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	BorgOriginal.sol#L311,312,313,321,322,323,324,325,326,327,330,331,332,333,334
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 public name = "Borg Original"
string public symbol = "BORG"
uint8 public decimals = 18
uint256 public reflectionFee1 = 4
uint256 public reflectionFee2 = 4
uint256 public marketingFee = 4
uint256 public treasuryFee = 4
uint256 public liquidityFee = 3
uint256 public burnFee = 1
uint256 public sellTax = 20
address public marketingWallet =
0x8534C63B9856Fe2Ed93aDf507600332B244A2fdC
address public treasuryWallet =
0x8136100798fdE72de63166a328eaE6aBE7368612
address public reflectionAddress1 =
0x8534C63B9856Fe2Ed93aDf507600332B244A2fdC
address public reflectionAddress2 =
0x8136100798fdE72de63166a328eaE6aBE7368612

...
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	BorgOriginal.sol#L63,73,83
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 _nonReentrantBefore() private {
        // On the first call to nonReentrant, _status will be
NOT_ENTERED
        if (_status == ENTERED) {
            revert ReentrancyGuardReentrantCall();
        }

...
    }

function _nonReentrantAfter() private {
        // By storing the original value once again, a refund
is triggered (see
        // https://eips.ethereum.org/EIPS/eip-2200)
        _status = NOT_ENTERED;
    }

...
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	BorgOriginal.sol#L93,120,222
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.

L22 - Potential Locked Ether

Criticality	Minor / Informative
Location	BorgOriginal.sol#L358
Status	Unresolved

Description

The contract contains Ether that has been placed into a Solidity contract and is unable to be transferred. Thus, it is impossible to access the locked Ether. This may produce a financial loss for the users that have called the payable method.

```
receive() external payable {
        // Process the received Ether
        // You can perform actions here
    }
```

Recommendation

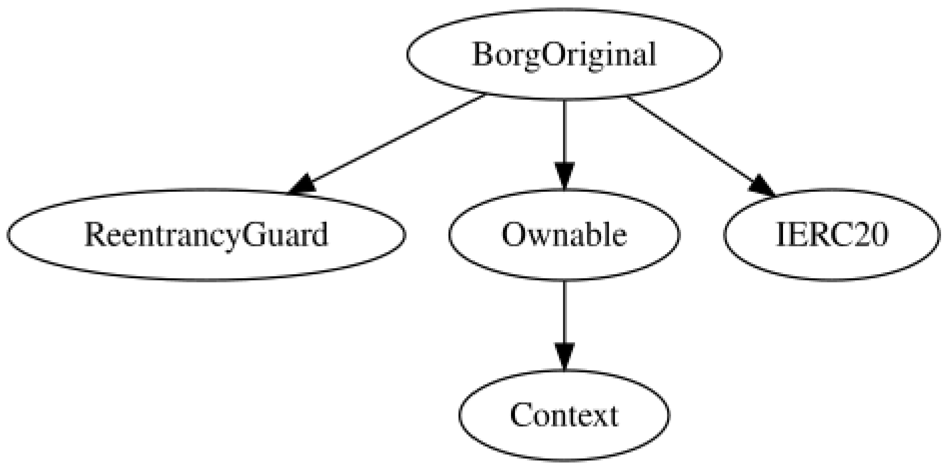
The team is advised to either remove the payable method or add a withdraw functionality. it is important to carefully consider the risks and potential issues associated with locked Ether.

Functions Analysis

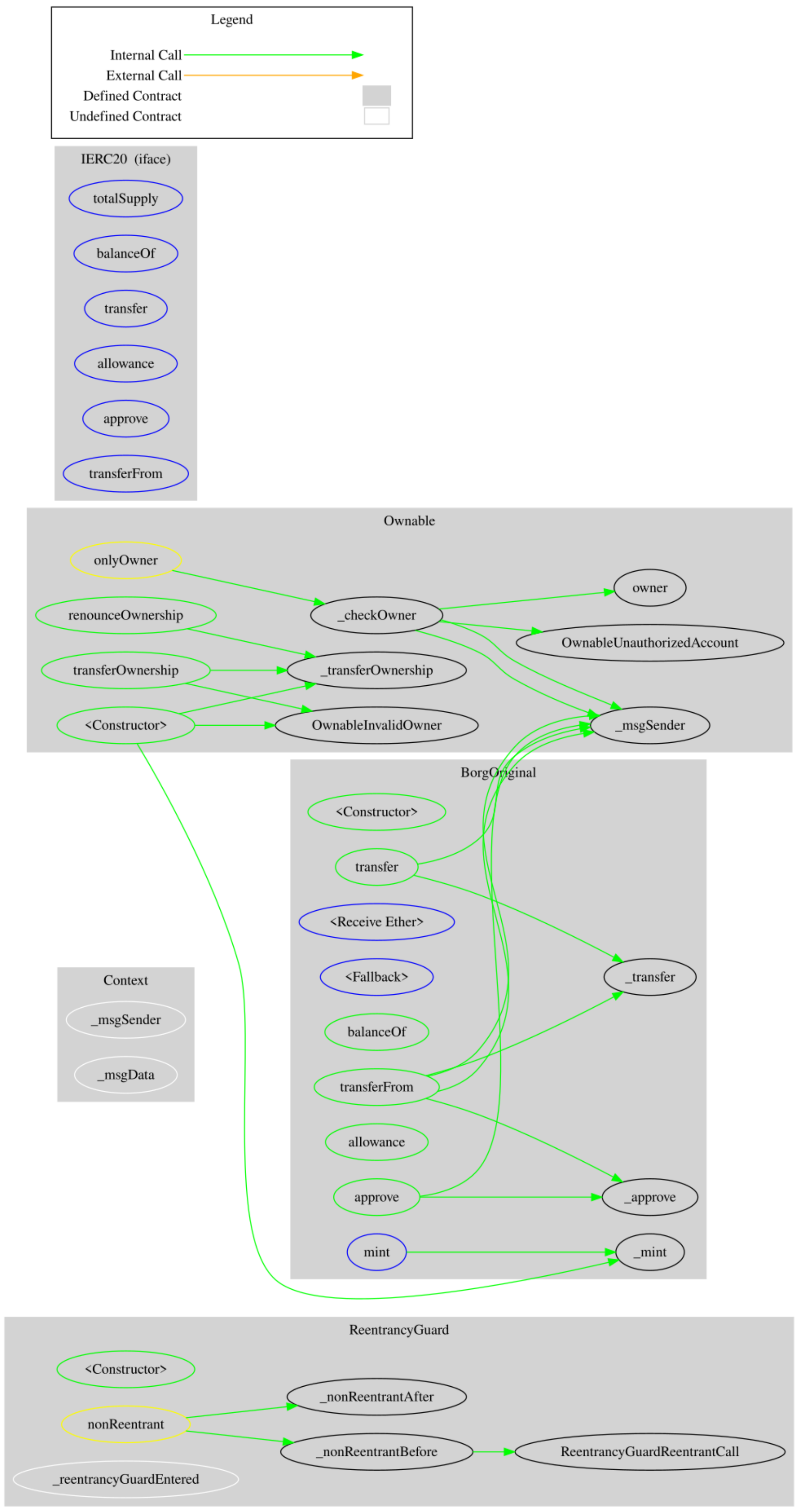
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
ReentrancyGuard	Implementation			
		Public	✓	-
	_nonReentrantBefore	Private	✓	
	_nonReentrantAfter	Private	✓	
	_reentrancyGuardEntered	Internal		
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	

IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
BorgOriginal	Implementation	IERC20, Ownable, ReentrancyGuard		
		Public	✓	Ownable
	_mint	Internal	✓	
	mint	External	✓	onlyOwner
		External	Payable	-
		External	✓	-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	_approve	Internal	✓	
	_transfer	Internal	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Ownable
onlyOwner
renounceOwnership
transferOwnership
<Constructor>
_checkOwner
_transferOwnership
OwnableInvalidOwner
owner
OwnableUnauthorizedAccount
_msgSender
BorgOriginal
<Constructor>
transfer
<Receive Ether>
<Fallback>
balanceOf
transferFrom
allowance
approve
mint
_transfer
_approve
_mint
Context
_msgSender
_msgData
ReentrancyGuard
<Constructor>
nonReentrant
_reentrancyGuardEntered
_nonReentrantAfter
_nonReentrantBefore
ReentrancyGuardReentrantCall

Summary

Borg Original 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 manipulate the fees and mint tokens. If the contract owner abuses the mint functionality, then the contract will be highly inflated. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats.

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