Cyberscope

Audit Report

Borg Original

November 2023

SHA256 6352c7da47f2a91c076b0aac0eda8bfbd733bb6bc0bd41ea33e5727b6c0a13ac

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
●	TSD	Total Supply Diversion	Unresolved
●	UPA	Unexcluded Pinksale Address	Unresolved
●	UAV	Uninitialized Addresses Variables	Unresolved
●	RVD	Redundant Variable Declaration	Unresolved
●	MTI	Misleading Tax Implementation	Unresolved
●	RRC	Redundant Require Check	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved

Table of Contents

Review

Contract Name	BorgOriginal
Testing Deploy	https://testnet.bscscan.com/address/0x090db0c2752b9644d70acca6e1a8a0498a0e1a27
Symbol	BORG
Decimals	18
Total Supply	70,000,000,000,000

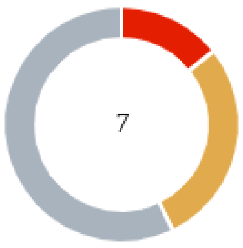
Audit Updates

Initial Audit	25 Oct 2023 https://github.com/cyberscope-io/audits/blob/main/borg/v1/audit.pdf
Corrected Phase 2	02 Nov 2023

Source Files

Filename	SHA256
BorgOriginal.sol	6352c7da47f2a91c076b0aac0eda8bfbd733bb6bc0bd41ea33e5727b6c0a13ac

Findings Breakdown



●	Critical	1
●	Medium	2
●	Minor / Informative	4

Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	1	0	0	0
●	Medium	2	0	0	0
●	Minor / Informative	4	0	0	0

TSD - Total Supply Diversion

Criticality	Critical
Location	BorgOriginal.sol#L334
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.

Specifically, the contract allocates various fees, including the `treasuryFeeShare` , from the total transaction amount. However, the contract attempts to calculate the `treasuryFeeShare` variable by subtracting only the `reflectionFee1Share` , `reflectionFee2Share` , and `marketingFeeShare` variables from the total amount. This approach does not account for all the fees and the `_balances` of the recipient. As a result, the balance of the `treasuryFeeShare` will be set to an incorrect number.

Additionally when the recipient is the owner, the contract calculates a `netAmount` based on the `sellTax` variable. However, despite this special calculation for the owner, the contract proceeds to calculate and distribute fees based on individual fee variables (`reflectionFee1` , `reflectionFee2` , `marketingFee` , `treasuryFee` , `liquidityFee` , and `burnFee`) rather than using the previously calculated `sellTax` . This inconsistency can lead to discrepancies in fee distribution when the recipient is the owner, as the fees applied do not align with the intended `sellTax` .

```
if (recipient == owner()) {
    netAmount = (amount * sellTax) / 100;
    totalFee = sellTax;
} else {
    totalFee = reflectionFee1 + reflectionFee2 + marketingFee +
treasuryFee + liquidityFee + burnFee;
    netAmount = (amount * totalFee) / 100;
}


// Allocate liquidityFee and burnFee to their respective destinations
_balances[liquidityWallet] = _balances[liquidityWallet] + (amount *
liquidityFee) / 100;
_balances[burnWallet] = _balances[burnWallet] + (amount * burnFee) /
100;
// Deduct the fee and send the net amount
    _balances[sender] = _balances[sender] - amount;
    _balances[recipient] = _balances[recipient] + (amount - netAmount);

// Calculate all fees first to mitigate rounding errors
uint256 reflectionFee1Share = (amount * reflectionFee1) / 100;
uint256 reflectionFee2Share = (amount * reflectionFee2) / 100;
uint256 marketingFeeShare = (amount * marketingFee) / 100;
uint256 treasuryFeeShare = amount - (reflectionFee1Share +
reflectionFee2Share + marketingFeeShare);
```

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 reconsider the calculation of the `treasuryFeeShare` fee. The fee should be calculated as the total amount minus the sum of all the fees that have occurred, plus the balance of the recipient. This will ensure that the `treasuryFeeShare` accurately represents the remaining fee after all other fees and balances have been accounted for. Additionally, it is recommended to use the `sellTax` variable to distribute the tax when the `recipient` is the owner.

UPA - Unexcluded Pinksale Address

Criticality	Medium
Location	BorgOriginal.sol#L315
Status	Unresolved

Description

The contract is designed with a fee mechanism that applies to each transaction. This design poses a significant obstacle to integration with platforms like Pinksale. Specifically, for the creation of a launchpad on Pinksale, the Pinksale factory address must be exempted from these transaction fees. Without this exemption, the creation of the pool on Pinksale will be prevented.

```
if (recipient == owner()) {
    netAmount = (amount * sellTax) / 100;
    totalFee = sellTax;
} else {
    totalFee = reflectionFee1 + reflectionFee2 + marketingFee +
treasuryFee + liquidityFee + burnFee;
    netAmount = (amount * totalFee) / 100;
}
```

Recommendation

It is recommended to modify the contract to exclude the Pinksale factory address from the fee mechanism. This will ensure compatibility with Pinksale and facilitate the smooth creation of pools on the platform.

UAV - Uninitialized Addresses Variables

Criticality	Medium
Location	BorgOriginal.sol#L244
Status	Unresolved

Description

The contract contains the `reflectionAddress1` , `liquidityWallet` , `burnWallet` and `gameAddress` public address variables. These addresses are intended to be set after the deployment of the contract as indicated by the comment line. However, the contract does not provide any mechanism or function to set or update these addresses post-deployment. As a result, these addresses will be initialized with the zero address by default, and due to the absence of setter functions, they will remain unchanged. This means that any fees sent to these addresses will be sent to the zero address, leading to potential unintended behavior.

```
address public reflectionAddress1; // To be after deployment
address public liquidityWallet; // To be after deployment
address public burnWallet; //To be after deployment
address public gameAddress; // To be set for borgBLOCK
```

Recommendation

It is recommended to provide initialization functions or setters for these address variables. This will allow the contract owner to set or update the addresses post-deployment, ensuring that the contract operates as intended and avoids potential unintended transactions associated with the zero address.

RVD - Redundant Variable Declaration

Criticality	Minor / Informative
Location	BorgOriginal.sol#L248
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 the `gameAddress` variable that is not used in a meaningful way by the contract. As a result, the variable is redundant.

```
address public gameAddress; // To be set for borgBLOCK
```

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.

MTI - Misleading Tax Implementation

Criticality	Minor / Informative
Location	BorgOriginal.sol#L315
Status	Unresolved

Description

The contract is designed to levy a tax when a transfer qualifies as a sale, as indicated by the sellTax variable. However, the actual implementation within the if statement doesn't specifically target sell transactions. Rather than applying the `sellTax` solely to sales, the contract imposes this tax on every transfer where the `recipient` matches the owner's address. This inconsistency in applying the `sellTax` exclusively when the `owner` is the `recipient` might result in misunderstandings and unforeseen financial implications for those engaging with the contract.

```
if (recipient == owner()) {
    netAmount = (amount * sellTax) / 100;
    totalFee = sellTax;
} else {
    totalFee = reflectionFee1 + reflectionFee2 + marketingFee +
treasuryFee + liquidityFee + burnFee; // 4 4 4 4 3 1 = 20
    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 the owner's address.

RRC - Redundant Require Check

Criticality	Minor / Informative
Location	BorgOriginal.sol#L311
Status	Unresolved

Description

The contract contains the `totalFee` variable which is initialized to zero just before a `require` statement that checks if `totalFee` is less than or equal to `25`. Given that `totalFee` is always set to zero at this point in the code, the `require` statement will always pass, rendering it redundant and ineffective in its intended purpose.

```
uint256 totalFee = 0;
require(totalFee <= 25, 'Total fee should not exceed 25%');
```

Recommendation

It is recommended to remove the `require` check from the contract since it does not provide any meaningful validation or protection against potential misuse or errors.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	BorgOriginal.sol#L244,246,247,248
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.

```
address public reflectionAddress1
address public liquidityWallet
address public burnWallet
address public gameAddress
```

Recommendation

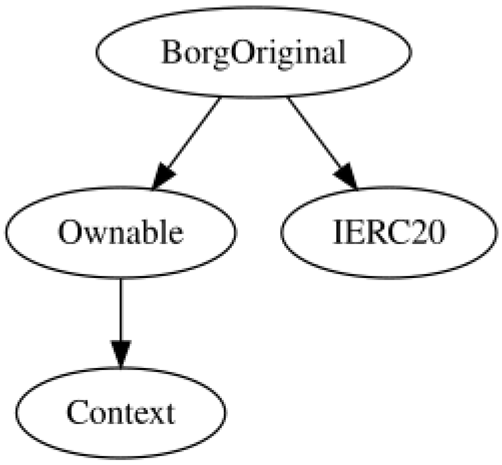
Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

Functions Analysis

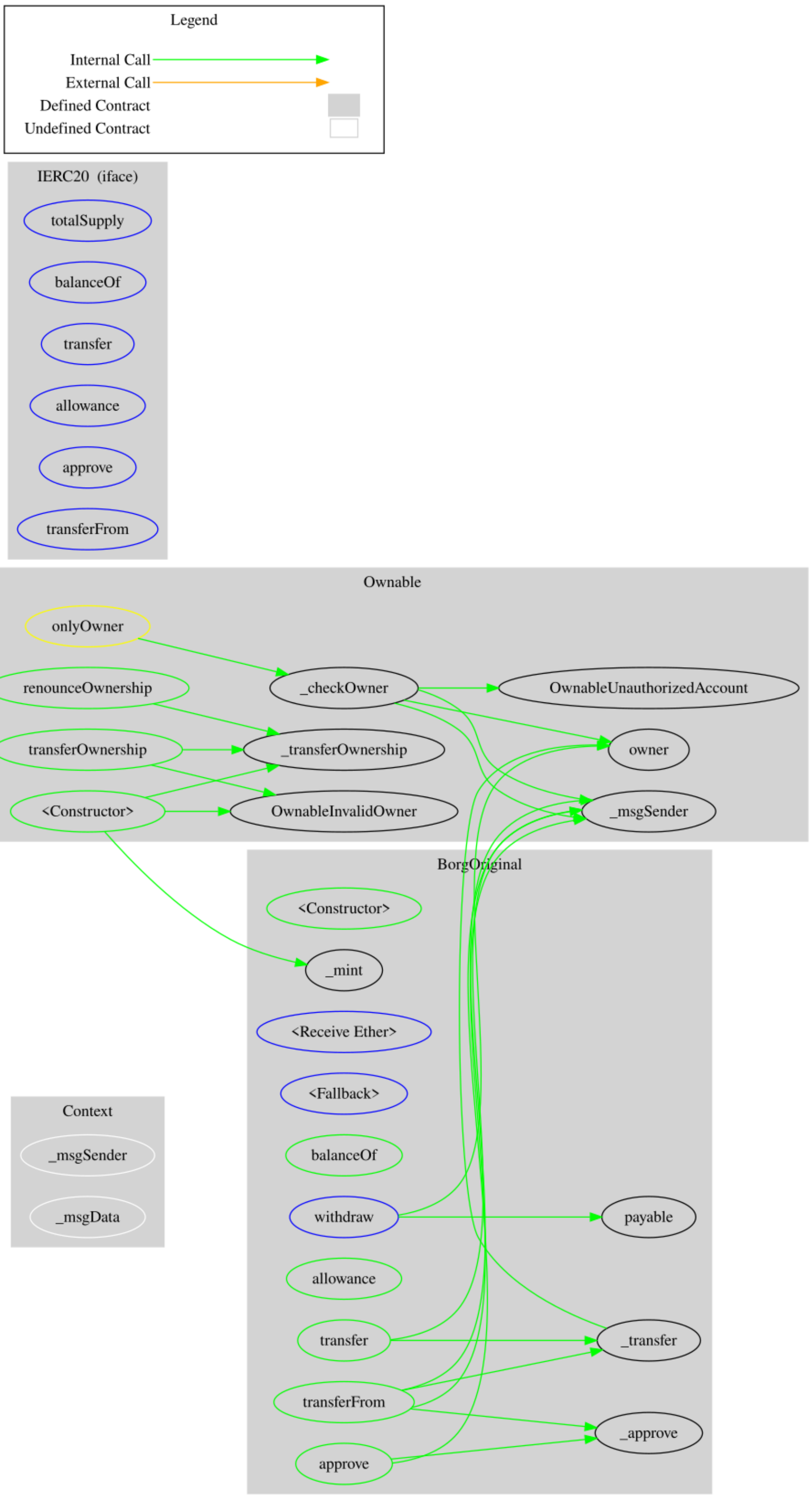
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-

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

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
OwnableUnauthorizedAccount
owner
_msgSender
BorgOriginal
<Constructor>
_mint
<Receive Ether>
<Fallback>
balanceOf
withdraw
allowance
transfer
transferFrom
approve
payable
_transfer
_approve
Context
_msgSender
_msgData

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 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. There is also a limit of max 20% 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