Cyberscope

Audit Report

Orina

October 2024

Network BSC

Address 0x4B09f251BaFa8a128B9e31e665A9E204184eB271

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
	IDI	Immutable Declaration Improvement	Unresolved
	MEM	Missing Error Messages	Unresolved
	MPM	Missing Pause Modifier	Unresolved
	OBF	Owner-Restricted Burn Function	Unresolved
	RSML	Redundant SafeMath Library	Unresolved
	RSRS	Redundant SafeMath Require Statement	Unresolved
	RWF	Redundant Withdrawal Function	Unresolved
	UTDE	Unverified Third-Party Dependencies Execution	Unresolved
	ZATR	Zero Amount Transfer Restriction	Unresolved
	L02	State Variables could be Declared Constant	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L15	Local Scope Variable Shadowing	Unresolved
	L17	Usage of Solidity Assembly	Unresolved
	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
Critical	Highly Likely / High Impact
Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
Minor / Informative	Unlikely / Low to no Impact

Review

Contract Name	Orina
Compiler Version	v0.8.8+commit.dddeac2f
Optimization	200 runs
Explorer	https://bscscan.com/address/0x4b09f251bafa8a128b9e31e665a9e204184eb271
Address	0x4b09f251bafa8a128b9e31e665a9e204184eb271
Network	BSC
Symbol	ORI
Decimals	18
Total Supply	1,000,000,000
Badge Eligibility	Yes

Audit Updates

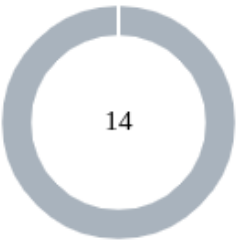
Initial Audit	11 Oct 2024

Source Files

Filename	SHA256
StandardToken.sol	a4910b16ccbc0f49d004699feaea6ad15d1b62919539834a6dce2955b732232e
SmartToken.sol	15fc7f98cd1938bb381da76fb8f81d40c9f691425d15aad6c38045da5590b7a9

Pauseable.sol	f04432f526248f1108f6483a5ca6298dfc7a1549fb138a17ae821583e3a7fa3f
Ownable.sol	8f544b32a703ed7bd4786f0727f9825a679dbbb17186d646b46b31387261a745
Orina.sol	4e739a12712cab0f69e516979bfedfa3fc97c03580a8dcab94abfc388d5d3500
BasicToken.sol	006bd91ae9b277f079667abbee44a26b0535ea0b60472ae0ea4b5580148f4301

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

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	Orina.sol#L15
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.

```
_decimals
```

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.

MEM - Missing Error Messages

Criticality	Minor / Informative
Location	StandardToken.sol#L30,64 Pauseable.sol#L20 Ownable.sol#L17,22 Orina.sol#L51 BasicToken.sol#L57,58,59,78
Status	Unresolved

Description

The contract is missing error messages. Specifically, there are no error messages to accurately reflect the problem, making it difficult to identify and fix the issue. As a result, the users will not be able to find the root cause of the error.

```
require(_allowed[from][msg.sender] >= value)
require(amount > 0 && _balances[msg.sender] >= amount)
require(!stopped)
require(msg.sender == _owner)
require(_newOwner != address(0))
require(Frozen[to] >= value)
require(from != address(0))
require(value > 0)
require(_balances[from].sub(Frozen[from]) >= value)
require(_recipient != address(0) && _recipient !=
address(this))
```

Recommendation

The team is suggested to provide a descriptive message to the errors. This message can be used to provide additional context about the error that occurred or to explain why the contract execution was halted. This can be useful for debugging and for providing more information to users that interact with the contract.

MPM - Missing Pause Modifier

Criticality	Minor / Informative
Location	SmartToken.sol#L18 BasicToken.sol#L51 StandardToken.sol#L29
Status	Unresolved

Description

The contract implements a `stoppable` modifier to prevent transactions when the contract is paused. However, the `transferAndCall` function does not include this modifier, allowing users to execute token transfers even when the contract is paused. This inconsistency in applying the pause mechanism creates a loophole where transfers can bypass the intended transaction halt, undermining the purpose of the pause functionality.

Notably, the current state of the contract shows that the owner has already renounced ownership, and the contract is not in a paused state. As a result, the owner cannot pause the contract, making this finding less impactful under current conditions, but still a potential issue if ownership is restored or in a different deployment.

```
    function transferAndCall(address _to, uint256 _value, bytes memory
_data) public override validRecipient(_to) returns(bool success) {
        _transfer(msg.sender, _to, _value);
        emit Transfer(msg.sender, _to, _value, _data);
        if (isContract(_to)) {
            contractFallback(_to, _value, _data);
        }
        return true;
    }

    function contractFallback(address _to, uint _value, bytes memory
_data) private {
        BEP677Receiver receiver = BEP677Receiver(_to);
        receiver.onTokenTransfer(msg.sender, _value, _data);
    }

    function  transfer(address to, uint256 value) public override
stoppable validRecipient(to) returns(bool) {
        _transfer(msg.sender, to, value);
        return true;
    }

    function transferFrom(address from, address to, uint256 value)
public override stoppable validRecipient(to) returns(bool) {
        ...
        return true;
    }
```

Recommendation

It is recommended to add the `stoppable` modifier to the `transferAndCall` function to ensure that all transfers are properly restricted when the contract is paused. This will maintain consistency in enforcing the pause mechanism across all relevant functions, preventing unauthorized transfers during paused periods, especially if ownership is regained in the future.

OBF - Owner-Restricted Burn Function

Criticality	Minor / Informative
Location	StandardToken.sol#L63
Status	Unresolved

Description

The contract is designed to allow the `burn` function, which takes `msg.sender` as the account to burn tokens from. However, the function is restricted by the `onlyOwner` modifier, meaning that only the contract owner can call it. This could lead to confusion if the intent was to allow any user to burn their own tokens, as using `msg.sender` suggests. If the goal is to let any user burn tokens from their own balance, the restriction to the owner is unnecessary and could limit functionality.

```
   function burn(uint256 amount) public stoppable onlyOwner
returns(bool) {
        require(amount > 0 && _balances[msg.sender] >= amount);
        totalSupply = totalSupply.sub(amount);
        _balances[msg.sender] =
_balances[msg.sender].sub(amount);
        emit Transfer(msg.sender, address(0), amount);
        return true;
    }
```

Recommendation

It is recommended to review the intended design of the `burn` function. If the intent is to allow any user to burn their own tokens, the `onlyOwner` modifier should be removed, enabling broader access to the function. Alternatively, if the burn functionality is intentionally restricted to the owner, the implementation and documentation should clarify this to avoid confusion.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	StandardToken.sol Orina.sol BasicToken.sol
Status	Unresolved

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert to underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases gas consumption unnecessarily in cases where the explanatory error message is not used.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library in cases where the revert error message is not used. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on https://docs.soliditylang.org/en/stable/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

RSRS - Redundant SafeMath Require Statement

Criticality	Minor / Informative
Location	BasicToken.sol#L9
Status	Unresolved

Description

The contract utilizes a `require` statement within the `add` function aiming to prevent overflow errors. This function is designed based on the SafeMath library's principles. In Solidity version 0.8.0 and later, arithmetic operations revert on overflow and underflow, making the overflow check within the function redundant. This redundancy could lead to extra gas costs and increased complexity without providing additional security.

```
    function add(uint256 a, uint256 b) internal pure returns
(uint256) {
        uint256 c = a + b;
        require(c >= a);
        return c;
    }
```

Recommendation

It is recommended to remove the `require` statement from the add function since the contract is using a Solidity pragma version equal to or greater than 0.8.0. By doing so, the contract will leverage the built-in overflow and underflow checks provided by the Solidity language itself, simplifying the code and reducing gas consumption. This change will uphold the contract's integrity in handling arithmetic operations while optimizing for efficiency and cost-effectiveness.

RWF - Redundant Withdrawal Function

Criticality	Minor / Informative
Location	Orina.sol#L60
Status	Unresolved

Description

The contract is designed without any functionality to accept funds, as it lacks a `receive()` , `payable fallback()` , or any `payable` functions. This means that no native currency (such as BNB or ETH) can be transferred or stored in the contract. However, the contract includes a `withdrawBNB` function that attempts to transfer the contract's balance to the owner. Since the contract is not capable of receiving or holding BNB, this function becomes redundant and ineffective. This redundancy could lead to confusion, as users or auditors might mistakenly believe that the contract can hold and withdraw native currency, which is not the case.

```
    function withdrawBNB() public onlyOwner returns(bool) {
         payable(msg.sender).transfer(address(this).balance);
         return true;
     }
```

Recommendation

It is recommended to remove the `withdrawBNB` function from the contract, as it serves no purpose in a contract that is not designed to accept or hold funds. This will simplify the code and prevent unnecessary complexity or misunderstandings regarding the contract’s ability to handle native currency. Additionally, it will improve clarity about the contract’s intended functionality.

UTDE - Unverified Third-Party Dependencies Execution

Criticality	Minor / Informative
Location	SmartToken.sol#L18
Status	Unresolved

Description

The contract uses an external contract in order to determine the transaction's flow. The external contract is untrusted. As a result, it may produce security issues and harm the transactions.

The contract allows for the execution of external contract logic during token transfers via the `transferAndCall` function. Specifically, when tokens are sent to a contract address, the contract invokes the `contractFallback` function, which in turn calls the `onTokenTransfer` function of the recipient contract. Since the external contract (the recipient of the tokens) can be untrusted, this introduces the potential for security risks. The external contract may execute malicious code within the `onTokenTransfer` function, leading to possible reentrancy attacks, unexpected behavior, or manipulation of the transaction flow. Additionally, the contract does not impose any limitations or verification on the code executed in the external contract, which could harm the security and reliability of the token transfers.

```
    function transferAndCall(address _to, uint256 _value, bytes
memory _data) public override validRecipient(_to) returns(bool
success) {
        _transfer(msg.sender, _to, _value);
        emit Transfer(msg.sender, _to, _value, _data);
        if (isContract(_to)) {
            contractFallback(_to, _value, _data);
        }
        return true;
    }

    function contractFallback(address _to, uint _value, bytes
memory _data) private {
        BEP677Receiver receiver = BEP677Receiver(_to);
        receiver.onTokenTransfer(msg.sender, _value, _data);
    }
```

Recommendation

The contract should use a trusted external source. A trusted source could be either a commonly recognized or an audited contract. The pointing addresses should not be able to change after the initialization. It is recommended to implement checks and safeguards to limit the potential risks of interacting with untrusted external contracts. Consider verifying the integrity or trustworthiness of the external contract before allowing it to execute logic within the `onTokenTransfer` function. Additionally, include mechanisms like reentrancy guards and strict validation of external contract behavior to reduce the risk of malicious exploits.

ZATR - Zero Amount Transfer Restriction

Criticality	Minor / Informative
Location	BasicToken.sol#L56
Status	Unresolved

Description

The contract is designed to prevent the transfer of zero token amounts, as enforced by the `_transfer` function, which includes a requirement that the `value` must be greater than zero. However, this behavior does not align with the ERC-20 standard, which explicitly allows for zero-value transfers. The ability to transfer zero tokens is sometimes used for signaling or for interacting with other contracts that expect ERC-20-compliant behavior, meaning this restriction may cause compatibility issues with other systems or dApps relying on the ERC-20 standard.

```
    function _transfer(address from, address to, uint256 value)
internal {
        require(from != address(0));
        require(value > 0);
        require(_balances[from].sub(Frozen[from]) >= value);
        _balances[from] = _balances[from].sub(value);
        _balances[to] = _balances[to].add(value);
        emit Transfer(from, to, value);
    }
```

Recommendation

It is recommended to modify the `_transfer` function to allow zero-value transfers, ensuring full compliance with the ERC-20 standard. This change will help ensure compatibility with external systems that expect the standard behavior and allow for flexibility in contract interactions that may involve zero-value transfers.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	BasicToken.sol#L37
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.

```
uint public totalSupply
```

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	StandardToken.sol#L36 SmartToken.sol#L18,27,32 Ownable.sol#L21 Orina.sol#L56 BasicToken.sol#L47,49,65,69,73
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 _spender
address _owner
address _to
bytes memory _data
uint256 _value
uint _value
address _addr
address _newOwner
uint256 _amount
address _tokenAddress
mapping(address => uint256) internal Frozen
mapping(address => uint256) internal _balances
```

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/stable/style-guide.html#naming-conventions.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	StandardToken.sol#L24,36 BasicToken.sol#L65,69,73
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.

```
address _owner
```

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.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	SmartToken.sol#L34
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 { length := extcodesize(_addr) }
```

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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	Orina.sol#L57
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.

```
IBEP20(_tokenAddress).transfer(_to, _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.

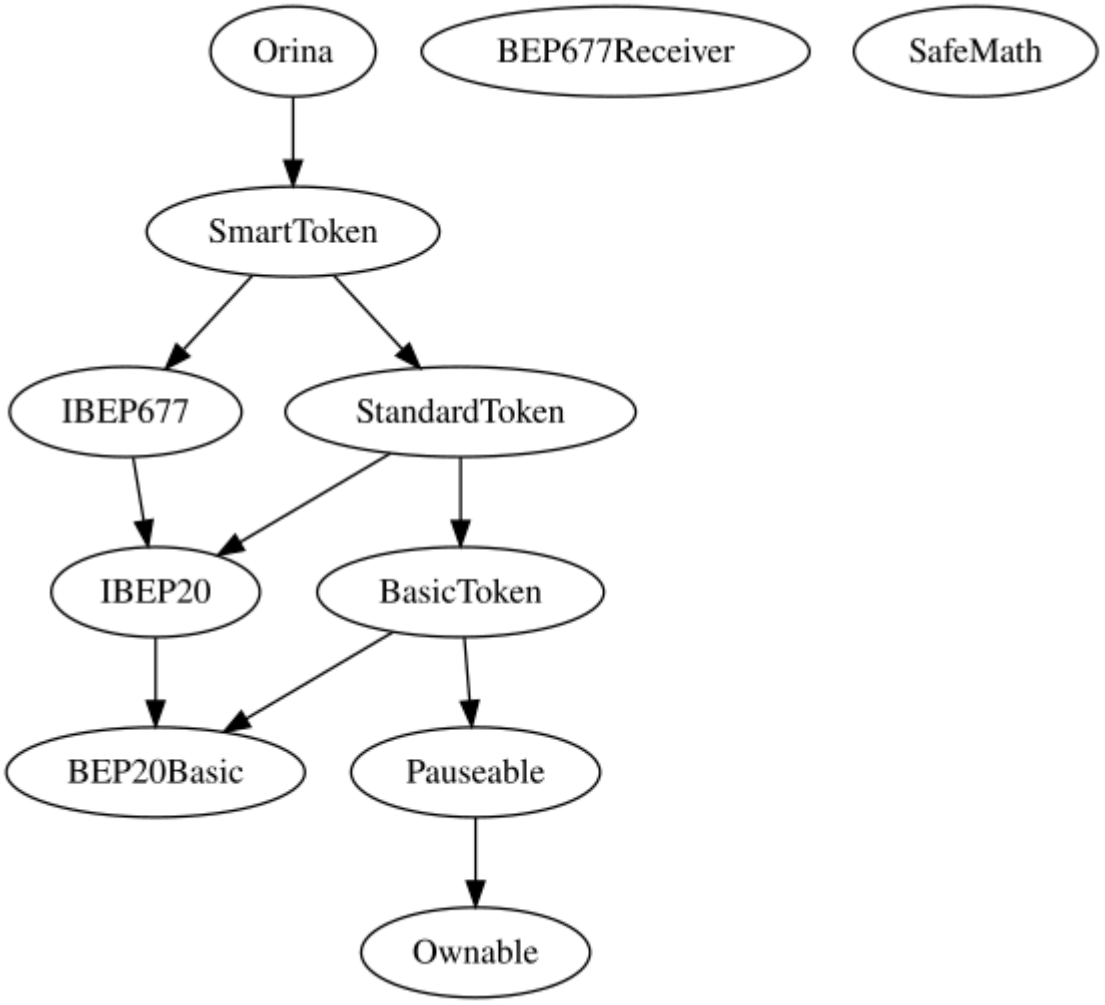
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
StandardToken	Implementation	IBEP20, BasicToken		
	approve	Public	✓	stoppable validRecipient
	_approve	Private	✓	
	transferFrom	Public	✓	stoppable validRecipient
	allowance	Public		-
	increaseAllowance	Public	✓	stoppable validRecipient
	decreaseAllowance	Public	✓	stoppable validRecipient
	mint	Public	✓	onlyOwner stoppable validRecipient
	burn	Public	✓	stoppable onlyOwner
SmartToken	Implementation	IBEP677, StandardToken		
	transferAndCall	Public	✓	validRecipient
	contractFallback	Private	✓	
	isContract	Private		
Pauseable	Implementation	Ownable		

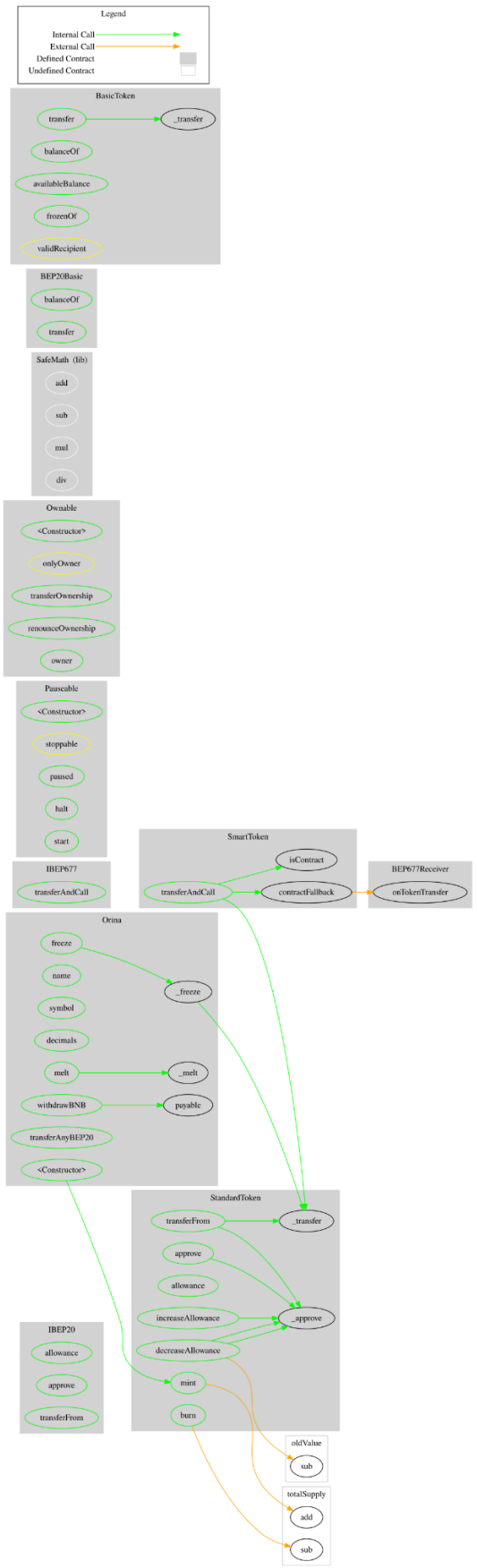
		Public	✓	-
	paused	Public		-
	halt	Public	✓	onlyOwner
	start	Public	✓	onlyOwner
Ownable	Implementation			
		Public	✓	-
	transferOwnership	Public	✓	onlyOwner
	renounceOwnership	Public	✓	onlyOwner
	owner	Public		-
Orina	Implementation	SmartToken		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	freeze	Public	✓	onlyOwner stoppable
	_freeze	Private	✓	
	melt	Public	✓	onlyOwner stoppable
	_melt	Private	✓	
	transferAnyBEP20	Public	✓	onlyOwner
	withdrawBNB	Public	✓	onlyOwner

BasicToken	Implementation	BEP20Basic, Pauseable		
	transfer	Public	✓	stoppable validRecipient
	_transfer	Internal	✓	
	balanceOf	Public		-
	availableBalance	Public		-
	frozenOf	Public		-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
BasicToken
transfer
_transfer
balanceOf
availableBalance
frozenOf
validRecipient
BEP20Basic
balanceOf
transfer
SafeMath (lib)
add
sub
mul
div
Ownable
<Constructor>
onlyOwner
transferOwnership
renounceOwnership
owner
Pausable
<Constructor>
stoppable
paused
halt
start
IBEP677
transferAndCall
SmartToken
isContract
transferAndCall
contractFallback
BEP677Receiver
onTokenTransfer
Orina
freeze
name
symbol
decimals
melt
withdrawBNB
transferAnyBEP20
<Constructor>
_freeze
_melt
payable
StandardToken
transferFrom
approve
allowance
increaseAllowance
decreaseAllowance
mint
burn
_transfer
_approve
IBEP20
allowance
approve
transferFrom
oldValue
sub
totalSupply
add
sub

Summary

Orina contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. Orina is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues.

The contract's ownership has been renounced. The information regarding the transaction can be accessed through the following link:

https://bscscan.com/tx/0xda500ba70168ecc1f5cd0ccb5652814a8679da96f8ee308747959b331f6f53eb

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

cyberscope.io