Cyberscope

Audit Report

BigFoot

July 2024

Network ETH

Address 0xd093AE84b764f716348aAf4542c68CD966AF73a1

Audited by © cyberscope

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	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
●	TEI	Transfers Execution Inability	Unresolved
●	MC	Missing Check	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	PAV	Pair Address Validation	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L18	Multiple Pragma Directives	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Review

Contract Name	BIG_FOOT
Compiler Version	v0.8.12+commit.f00d7308
Optimization	200 runs
Explorer	https://etherscan.io/address/0xd093ae84b764f716348aaf4542c68cd966af73a1
Address	0xd093ae84b764f716348aaf4542c68cd966af73a1
Network	ETH
Symbol	BFT
Decimals	18
Total Supply	999,999,999,999,999
Badge Eligibility	Must Fix Criticals

Audit Updates

Initial Audit	17 Jul 2024

Source Files

Filename	SHA256
contracts/Token.sol	bf05362b434051b12824c5640406240211bdef95136e3eaf79f0321a74d78135

Overview

The contract is no longer functional due to a critical flaw identified during the audit. This issue arises from the renouncement of ownership by the initial owner without executing the `setRule` function, which is essential for configuring vital parameters such as the Uniswap pair address and transfer limits. Consequently, the contract variables remain in their default states, with the Uniswap pair address set to the zero address and the owner also being the zero address.

With these conditions, the `_beforeTokenTransfer` function enforces that token transfers can only occur if either the sender or the recipient is the owner. Since the owner is now the zero address, this effectively means that tokens can only be transferred to the zero address, resulting in their burning. This constraint locks the total supply of tokens with the initial owner and prevents any other transfers, rendering the contract unusable for its intended purpose.

To address this issue, a new deployment of the contract is necessary. The new contract should ensure that all critical parameters are properly set through the `setRule` function before any ownership renouncement occurs. This will restore the contract's functionality, allowing it to operate as intended and support typical token transactions.

Findings Breakdown

Critical	2
Medium	0
Minor / Informative	8

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	2	0	0	0
Medium	0	0	0	0
Minor / Informative	8	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	contracts/Token.sol#L706
Status	Unresolved

Description

As described in detail in the `TEI` finding, transactions are essentially stopped permanently, except for token burning, which originates from the initial contract owner that holds the entire token supply.

Recommendation

The team should follow the recommendations provided in the `TEI` finding.

TEI - Transfers Execution Inability

Criticality	Critical
Location	contracts/Token.sol#L706
Status	Unresolved

Description

The smart contract faces a critical issue where the transfer functionality is severely restricted. This problem arises from the combination of the initial owner's renouncement of ownership and the fact that the `setRule` function was never executed post-deployment. As a result, the contract variables are set to their default states, with limited being `false`, `maxHoldingAmount` and `minHoldingAmount` both set to zero, and `uniswapV2Pair` set to the zero address. The `_beforeTokenTransfer` function includes a condition that checks if the `uniswapV2Pair` address is zero, in which case it requires either the sender or the recipient to be the owner. Given that the owner is now the zero address and transfers cannot originate from this address, the only permissible transactions are those where the recipient is the zero address. Consequently, the initial owner, who holds all the tokens, is the only entity capable of transferring tokens, and these transfers can only be made to the zero address, effectively burning the tokens. This limitation renders the contract non-functional for practical token transfers.

```
if (uniswapV2Pair == address(0)) {
      require(from == owner() || to == owner(), "trading is not
started");
      return;
    }
```

Recommendation

To resolve this issue, a new deployment of the contract is necessary. The new contract should ensure that the setRule function is called immediately after deployment to set the necessary parameters correctly before any renouncement of ownership.

MC - Missing Check

Criticality	Minor / Informative
Location	contracts/Token.sol#L687
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues. Specifically, the contract lacks validation that `minHoldingAmount` is less than `maxHoldingAmount` .

```
function setRule(
    bool _limited,
    address _uniswapV2Pair,
    uint256 _maxHoldingAmount,
    uint256 _minHoldingAmount
  ) external onlyOwner {
    limited = _limited;
    uniswapV2Pair = _uniswapV2Pair;
    maxHoldingAmount = _maxHoldingAmount;
    minHoldingAmount = _minHoldingAmount;
  }
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	contracts/Token.sol#L680,687
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.

```
function blacklist(address _address, bool _isBlacklisting)
    external
    onlyOwner
  {
    blacklists[_address] = _isBlacklisting;
  }

  function setRule(
    bool _limited,
    address _uniswapV2Pair,
    uint256 _maxHoldingAmount,
    uint256 _minHoldingAmount
  ) external onlyOwner {
    limited = _limited;
    uniswapV2Pair = _uniswapV2Pair;
    maxHoldingAmount = _maxHoldingAmount;
    minHoldingAmount = _minHoldingAmount;
  }
```

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.

PAV - Pair Address Validation

Criticality	Minor / Informative
Location	contracts/Token.sol#L687
Status	Unresolved

Description

The contract is missing address validation in the pair address argument. The absence of validation reveals a potential vulnerability, as it lacks proper checks to ensure the integrity and validity of the pair address provided as an argument. The pair address is a parameter used in certain methods of decentralized exchanges for functions like token swaps and liquidity provisions.

The absence of address validation in the pair address argument can introduce security risks and potential attacks. Without proper validation, if the owner's address is compromised, the contract may lead to unexpected behavior like loss of funds.

```
function setRule(
    bool _limited,
    address _uniswapV2Pair,
    uint256 _maxHoldingAmount,
    uint256 _minHoldingAmount
  ) external onlyOwner {
    limited = _limited;
    uniswapV2Pair = _uniswapV2Pair;
    maxHoldingAmount = _maxHoldingAmount;
    minHoldingAmount = _minHoldingAmount;
  }
```

Recommendation

To mitigate the risks associated with the absence of address validation in the pair address argument, it is recommended to implement comprehensive address validation mechanisms. A recommended approach could be to verify pair existence in the decentralized application. Prior to interacting with the pair address contract, perform checks to verify the existence and validity of the contract at the provided address. This can be achieved by querying the provider's contract or utilizing external libraries that provide contract verification services.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/Token.sol#L669,680,688,689,690,691
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.

```
contract BIG_FOOT is Ownable, ERC20 {
  bool public limited;
  uint256 public maxHoldingAmount;
  uint256 public minHoldingAmount;
  address public uniswapV2Pair;
  mapping(address => bool) public blacklists;
...
    }
  }

  function burn(uint256 value) external {
    _burn(msg.sender, value);
  }
}

...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.
Find more information on the Solidity documentation https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	contracts/Token.sol#L638
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 _beforeTokenTransfer(
    address from,
    address to,
    uint256 amount
  ) internal virtual {}
```

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.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	contracts/Token.sol#L676
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.

```
uint256 _totalSupply
```

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.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	contracts/Token.sol#L55,81,160,247,275,667
Status	Unresolved

Description

If the contract includes multiple conflicting pragma directives, it may produce unexpected errors. To avoid this, it's important to include the correct pragma directive at the top of the contract and to ensure that it is the only pragma directive included in the contract.

```
pragma solidity ^0.8.0;
```

Recommendation

It is important to include only one pragma directive at the top of the contract and to ensure that it accurately reflects the version of Solidity that the contract is written in.

By including all required compiler options and flags in a single pragma directive, the potential conflicts could be avoided and ensure that the contract can be compiled correctly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/Token.sol#L55,81,160,247,275,667
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.

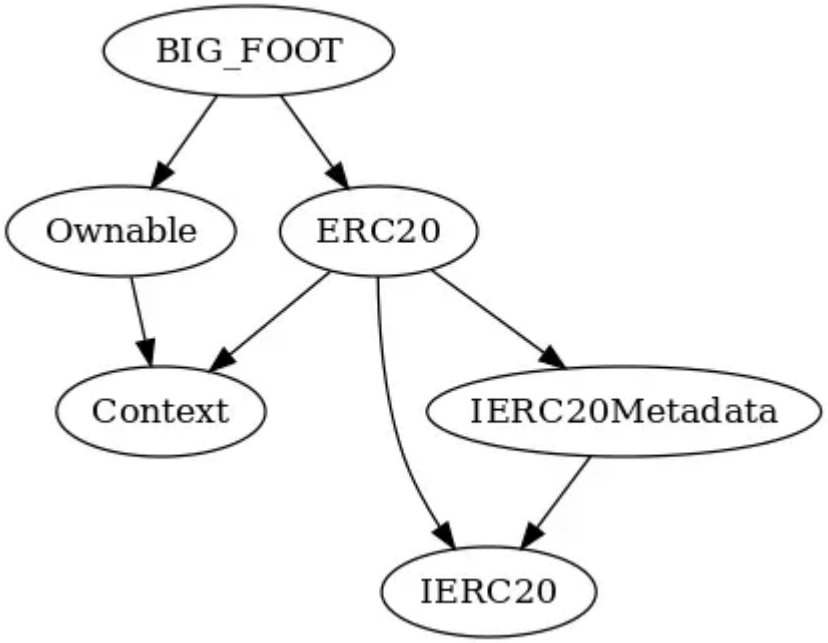
Functions Analysis

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

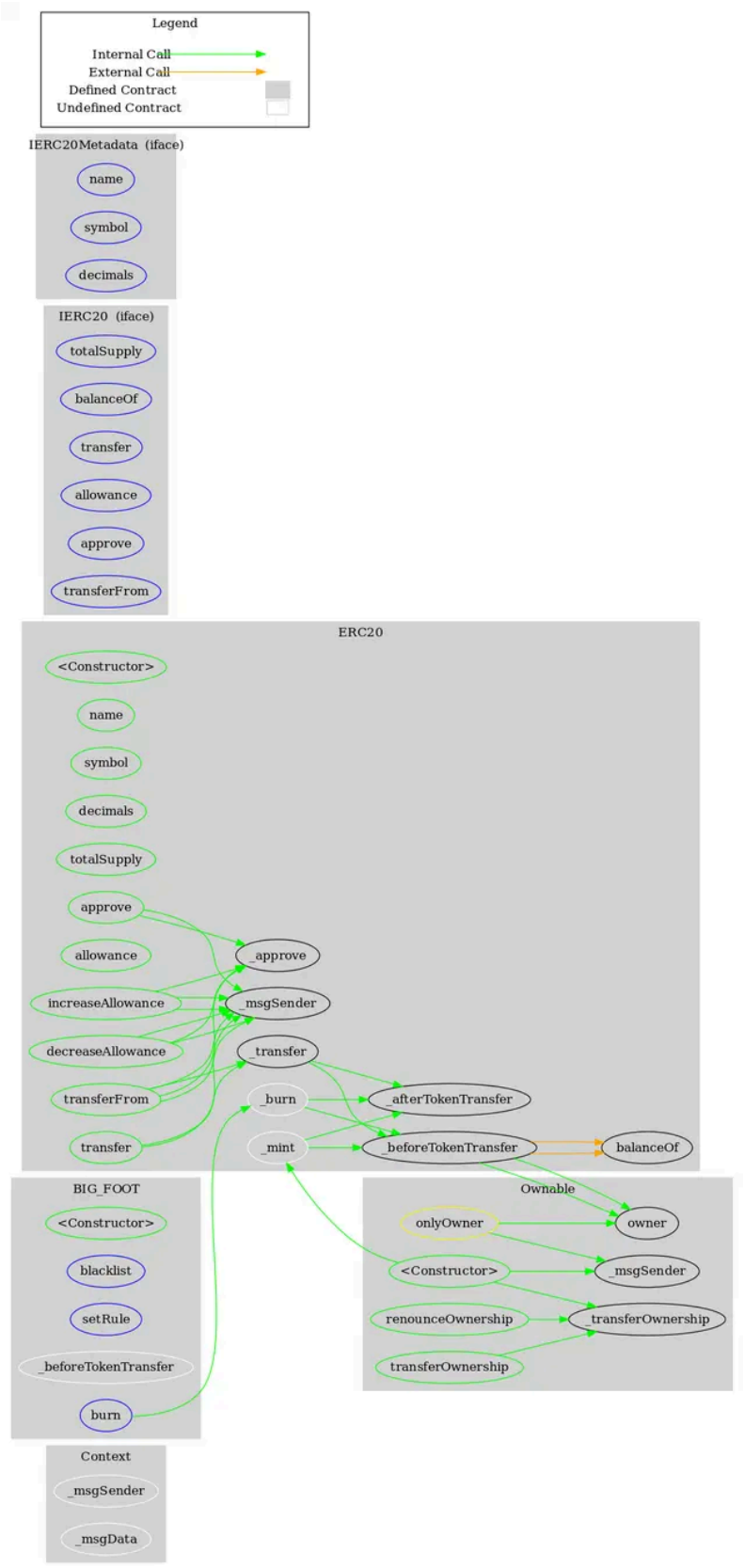
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	

	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
BIG_FOOT	Implementation	Ownable, ERC20		
		Public	✓	ERC20
	blacklist	External	✓	onlyOwner
	setRule	External	✓	onlyOwner
	_beforeTokenTransfer	Internal	✓	
	burn	External	✓	-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IERC20Metadata (iface)
name
symbol
decimals
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
ERC20
<Constructor>
name
symbol
decimals
totalSupply
approve
allowance
increaseAllowance
decreaseAllowance
transferFrom
transfer
_approve
_msgSender
_transfer
_burn
_mint
_afterTokenTransfer
_beforeTokenTransfer
balanceOf
BIG_FOOT
<Constructor>
blacklist
setRule
_beforeTokenTransfer
burn
Ownable
onlyOwner
owner
<Constructor>
_msgSender
renounceOwnership
_transferOwnership
transferOwnership
Context
_msgSender
_msgData

Summary

BigFoot contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. This audit reveals a critical flaw that renders the contract non-functional.

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