Cyberscope

Audit Report

Leaks

April 2024

SHA256 32806e141e220de562588b9e2dfe5ab5f6d7309efb2a07486fac2d89e79086ab

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
●	FLBV	Fees Logic Bypass Vulnerability	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	RFL	Redundant Fee Logic	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L22	Potential Locked Ether	Unresolved

Table of Contents

Review

Contract Name	LEAKS
Testing Deploy	https://testnet.bscscan.com/address/0x79656620bc9f4477f3a2b0e1dfe1079ed5636405
Symbol	LEAKS
Decimals	18
Total Supply	1,000,000
Badge Eligibility	Yes

Audit Updates

Initial Audit	01 Apr 2024

Source Files

Filename	SHA256
contracts/LEAKS.sol	32806e141e220de562588b9e2dfe5ab5f6d7309efb2a07486fac2d89e79086ab

Findings Breakdown

Critical	0
Medium	0
Minor / Informative	9

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	9	0	0	0

FLBV - Fees Logic Bypass Vulnerability

Criticality	Minor / Informative
Location	contracts/LEAKS.sol#L334
Status	Unresolved

Description

The contract is designed to apply a dynamic fee structure based on the number of transfers involving contract addresses, with specific milestones triggering changes in the tax rate and renouncing ownership when a certain threshold of contract transfers is reached. This mechanism is intended to regulate the flow of transactions and apply a tax based on the involvement of contract addresses. However, a vulnerability exists where a malicious user can exploit the contract by executing a large number of transactions with very small amounts in a short time frame. Since the logic for increasing the tax and eventually renouncing ownership is solely based on the count of contract transfers without considering the transaction amount or time intervals between transactions, an attacker can rapidly meet the threshold for ownership renunciation. This bypasses the intended economic disincentives of the fee structure and compromises the contract's ownership integrity, potentially leading to unintended consequences and loss of control over the contract's functions.

```
if (isContract(sender) || isContract(recipient)) {
      contractTransfersCount++;
      if (contractTransfersCount == 201) _renounceOwnership();
      else if (contractTransfersCount == 200) tax = 5;
      else if (contractTransfersCount == 100) tax = 10;
      else if (contractTransfersCount == 50) tax = 15;
      else if (contractTransfersCount == 20) tax = 20;
      else if (contractTransfersCount == 1) tax = 25;
      fee = (amount * tax) / 100;
```

Recommendation

It is recommended to add additional checks and reconsider the conditions under which renouncing ownership occurs to address the identified vulnerability. Specifically, the

contract should incorporate mechanisms to detect and prevent exploitation through rapid, minimal-value transactions. This could involve setting a minimum transaction value that qualifies for incrementing the contract transfer count, thereby ensuring that only transactions of a significant amount contribute to reaching the milestones for tax rate adjustments and ownership renunciation.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	contracts/LEAKS.sol#L314
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 setExcludedFromFees(address addr, bool state)
external onlyOwner {
        require(
            !excludedFromFees[addr],
            "This address already excluded form fees"
        );
        excludedFromFees[addr] = state;
    }
```

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.

RFL - Redundant Fee Logic

Criticality	Minor / Informative
Location	contracts/LEAKS.sol#L333
Status	Unresolved

Description

The contract is designed to apply a dynamic fee structure for transactions involving contracts, adjusting the tax rate based on the `contractTransfersCount`. This logic is intended to incrementally change until the count reaches `201`, at which point the ownership is renounced. However, the condition to check for contract involvement in a transfer continues to be evaluated for every transaction, even after reaching the `201` count threshold. This results in unnecessary checks and the application of a redundant logic that no longer affects the contract's state or behavior, potentially impacting the contract's efficiency and gas costs for transactions.

```
if (isContract(sender) || isContract(recipient)) {
    contractTransfersCount++;
    if (contractTransfersCount == 201) _renounceOwnership();
    else if (contractTransfersCount == 200) tax = 5;
    else if (contractTransfersCount == 100) tax = 10;
    else if (contractTransfersCount == 50) tax = 15;
    else if (contractTransfersCount == 20) tax = 20;
    else if (contractTransfersCount == 1) tax = 25;
    fee = (amount * tax) / 100;
```

Recommendation

It is recommended to introduce a condition to bypass the dynamic fee logic once the `contractTransfersCount` reaches or exceeds `201`. This could be achieved by adding a skip logic at the start of the fee calculation section if the count is greater than `201`, thereby optimizing contract performance and reducing unnecessary computational overhead.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contracts/LEAKS.sol#L292,296,297
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 private marketingWallet =
        0x2D1BA9DDBbFBfb1ac9E538105ff77280172A5dfD
string public _website = "https://t.me/vip_leaks_public"

string public _description =
        "Not Financial Advice. Always Do your own research."
```

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	contracts/LEAKS.sol#L296,297
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.

```
string public _website = "https://t.me/vip_leaks_public"

string public _description =
        "Not Financial Advice. Always Do your own research."
```

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/LEAKS.sol#L50,235
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 _contextSuffixLength() internal view virtual returns
(uint256) {
        return 0;
    }

...
```

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.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	contracts/LEAKS.sol#L319
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 {
            size := extcodesize(account)
        }
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/LEAKS.sol#L3
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 ^0.8.24;
```

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	contracts/LEAKS.sol#L352
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 {}
```

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.

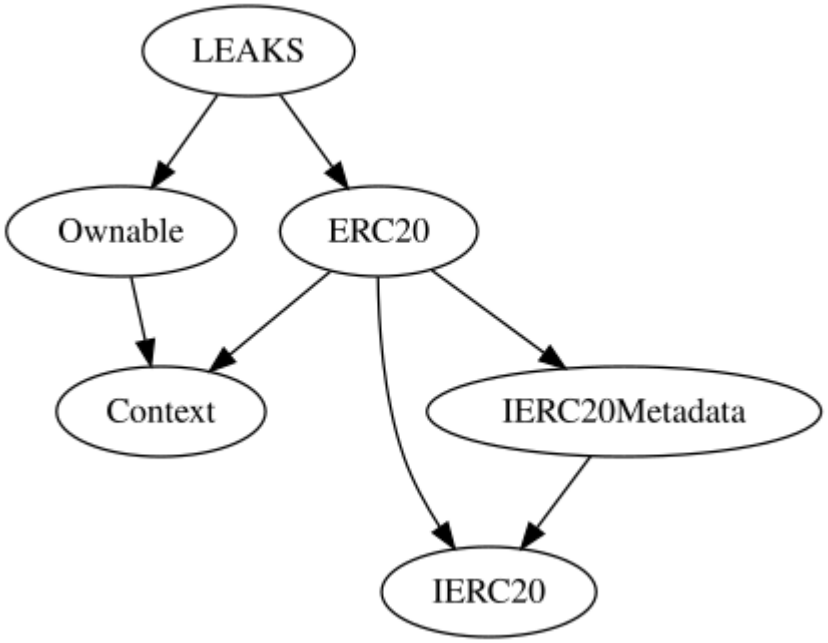
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
	_contextSuffixLength	Internal		
Ownable	Implementation	Context		
		Public	✓	-

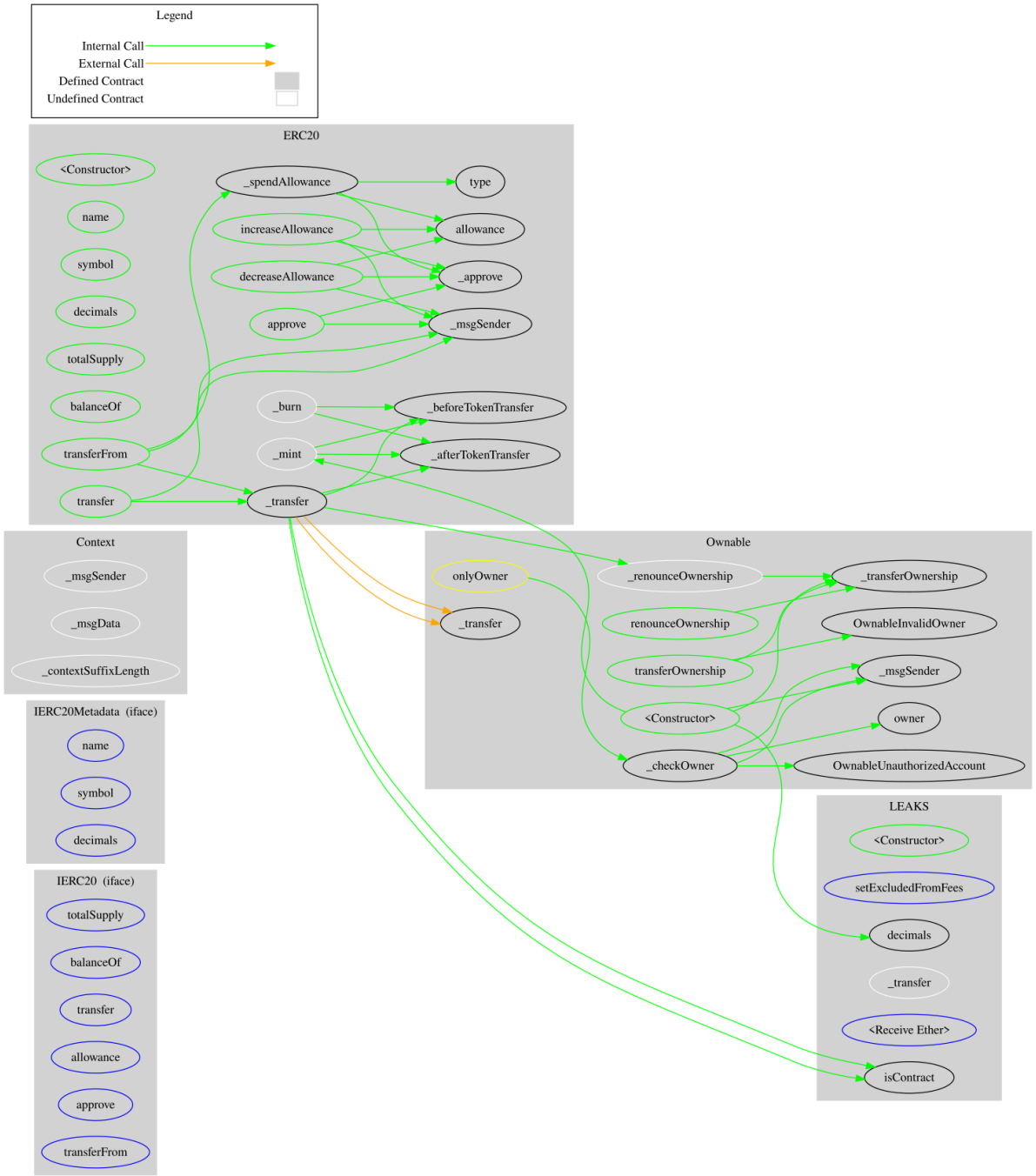
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	_renounceOwnership	Internal	✓	
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	

	_spendAllowance	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
LEAKS	Implementation	ERC20, Ownable		
		Public	✓	ERC20
	setExcludedFromFees	External	✓	onlyOwner
	isContract	Internal		
	_transfer	Internal	✓	
		External	Payable	-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
transferFrom
transfer
_spendAllowance
increaseAllowance
decreaseAllowance
approve
type
allowance
_approve
_msgSender
_burn
_mint
_beforeTokenTransfer
_afterTokenTransfer
_transfer
Context
_msgSender
_msgData
_contextSuffixLength
IERC20Metadata (iface)
name
symbol
decimals
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Ownable
onlyOwner
_transfer
_renounceOwnership
renounceOwnership
transferOwnership
<Constructor>
_checkOwner
_transferOwnership
OwnableInvalidOwner
_msgSender
owner
OwnableUnauthorizedAccount
LEAKS
<Constructor>
setExcludedFromFees
decimals
_transfer
<Receive Ether>
isContract

Summary

Leaks contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. leaks 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 a fixed fee of 2% for transactions, alongside a dynamic fee structure that is activated for transactions involving contracts.

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