Cyberscope

Audit Report

# Catpurr

February 2024

| | |
|---|---|
| Network | BSC |
| Address | 0x20bc80955b3893b012bc0fba3d1605de57e00c1c |
| Audited by | © cyberscope |

# Analysis

● Critical ● Medium ● Minor / Informative ● Pass

| Severity | Code | Description | Status |
|---|---|---|---|
| ● | ST | Stops Transactions | Acknowledged |
| ● | OTUT | Transfers User's Tokens | Passed |
| ● | ELFM | Exceeds Fees Limit | Unresolved |
| ● | MT | Mints Tokens | Passed |
| ● | BT | Burns Tokens | Passed |
| ● | BC | Blacklists Addresses | Passed |

## Diagnostics

● Critical ● Medium ● Minor / Informative

| Severity | Code | Description | Status |
|---|---|---|---|
| ● | US | Untrusted Source | Unresolved |
| ● | IEW | Inaccessible ETH Withdraw | Unresolved |
| ● | IEE | Inconsistent Event Emission | Unresolved |
| ● | PBV | Percentage Boundaries Validation | Unresolved |
| ● | RSW | Redundant Storage Writes | Unresolved |
| ● | L07 | Missing Events Arithmetic | Unresolved |
| ● | L09 | Dead Code Elimination | Unresolved |
| ● | L16 | Validate Variable Setters | Unresolved |
| ● | L18 | Multiple Pragma Directives | Unresolved |
| ● | L19 | Stable Compiler Version | Unresolved |

# Table of Contents

## Review

| | |
|---|---|
| **Contract Name** | CATPURR |
| **Compiler Version** | v0.8.17+commit.8df45f5f |
| **Optimization** | 200 runs |
| **Explorer** | https://bscscan.com/address/0x20bc80955b3893b012bc0fba3d1605de57e00c1c |
| **Address** | 0x20bc80955b3893b012bc0fba3d1605de57e00c1c |
| **Network** | BSC |
| **Symbol** | PURR |
| **Decimals** | 18 |
| **Total Supply** | 1,000,000,000,000 |
| **Badge Eligibility** | Must Fix Criticals |

| | |
|---|---|
| **Contract Name** | ExponentialTaxHandler |
| **Explorer** | https://bscscan.com/address/0x57fb5d58e28d1306fe96a6313ea19df0731c8be7 |

| | |
|---|---|
| **Contract Name** | TreasuryHandler |
| **Explorer** | https://bscscan.com/address/0x3e12d8d1246784acfcfdd50e68ed3b0ac5a59ccf |

## Audit Updates

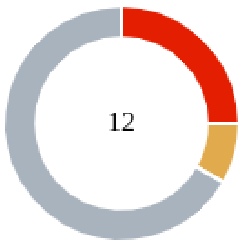
| | |
|---|---|
| **Initial Audit** | 19 Feb 2024<br>https://github.com/cyberscope-io/audits/blob/main/2-purr/v1/audit.pdf |
| **Corrected Phase 2** | 21 Feb 2024 |

## Source Files

| **Filename** | SHA256 |
|---|---|
| **contracts/Catpurr.sol** | a81f3461e46cd9071281b9a1fc023d3b5aa1097f92db83860b14c94eed5d12aa |
| **contracts/tax/ITaxHandler.sol** | 1861fb4ec6daa61d4d7f99cbe426e0a53f655b45b38352a0db9c35a64fe27883 |
| **contracts/treasury/TreasuryHandler.sol** | 98ee6b1bcedd4fe632f15bdf070ee662bc72cc874ecf5d0475dfc2178fdf2ad6 |
| **contracts/treasury/ITreasuryHandler.sol** | d802cb3e29064191f1e18f02220cc8181a09b4dda9aa4785385507808d85713d |
| **contracts/tax/ITaxHandler.sol** | 1861fb4ec6daa61d4d7f99cbe426e0a53f655b45b38352a0db9c35a64fe27883 |
| **contracts/interfaces/IERC20Burnable.sol** | 7d8240509cd52f429bc723fb9da3729cb3cb5bb8b396d5428fe4999cff561dab |
| **@openzeppelin/contracts/utils/Context.sol** | b2cfee351bcafd0f8f27c72d76c054df9b571b62cfac4781ed12c86354e2a56c |
| **@openzeppelin/contracts/token/ERC20/IERC20.sol** | 7ebde70853ccafcf1876900dad458f46eb9444d591d39bfc58e952e2582f5587 |
| **@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol** | af5c8a77965cc82c33b7ff844deb9826166689e55dc037a7f2f790d057811990 |

| | |
|---|---|
| **@openzeppelin/contracts/access/Ownable.sol** | a8e4e1ae19d9bd3e8b0a6d46577eec098c01fbaffd3ec1252fd20d799e73393b |

## Findings Breakdown



| | | |
|---|---|---|
| ● | Critical | 3 |
| ● | Medium | 1 |
| ● | Minor / Informative | 8 |

| Severity | | Unresolved | Acknowledged | Resolved | Other |
|---|---|---|---|---|---|
| ● | Critical | 2 | 1 | 0 | 0 |
| ● | Medium | 1 | 0 | 0 | 0 |
| ● | Minor / Informative | 8 | 0 | 0 | 0 |

## ST - Stops Transactions

| | |
|---|---|
| **Criticality** | Critical |
| **Location** | contracts/Catpurr.sol#L272 |
| **Status** | Acknowledged |

### Description

As described in the `US-Untrusted Source` and `ELFM-Exceeds Fees Limit` finding, the contract is utilizing the `treasuryHandler` contract, which the owner has the ability to change to a new external contract. This potential change to an external dependency poses the risk of arbitrarily halting sales transactions for all users except specific addresses. As a result, the contract may operate as a honeypot.

```
    function _transfer(address from, address to, uint256 amount) internal
virtual {
        ...
         treasuryHandler.beforeTransferHandler(from, to, amount);

         ...
         if (tax > 0) {
             address treasuryHandlerAdress = address(treasuryHandler);
             unchecked {
                 // Overflow not possible: the sum of all balances is
capped by totalSupply, and the sum is preserved by
                 _balances[treasuryHandlerAdress] += tax;
             }

             emit Transfer(from, treasuryHandlerAdress, tax);
         }

         treasuryHandler.afterTransferHandler(from, to, amount);

         emit Transfer(from, to, taxedAmount);
    }
```

## Recommendation

It is recommended to incorporate the transfer logic directly within the contract's codebase, rather than relying on an external contract. This change would significantly reduce the risk of unauthorized control and manipulation by centralizing the logic within the contract itself, thereby enhancing transparency and security. By embedding critical functionalities like the `beforeTransferHandler` and `afterTransferHandler` operations internally, the contract can ensure that all transactions adhere to predefined rules that are visible to everyone. Otherwise, consider renouncing the ownership of the `CATPURR` contract.

## Team Update

The team has acknowledged that this is not a security issue and states:

*1. The designated tax collection account is not a standard externally owned account (EOA); it functions through a specialized contract dedicated to tax management. Furthermore, this contract is managed under a Safe {WALLET} multisig wallet, controlled by a governance committee that mandates a minimum of four authorized signatures for any operational decisions.*

*2. Catpurr team will renounce the ownership for the contract and never claim back the privileged roles.*

## ELFM - Exceeds Fees Limit

| | |
|---|---|
| **Criticality** | Critical |
| **Location** | contracts/Catpurr.sol#L281<br>contracts/treasury/ExponentialTaxHandler#L33 |
| **Status** | Unresolved |

### Description

As described in the `US-Untrusted Source` finding the contract is utilizing the `taxHandler` external contract. This mechanism is used within the `_transfer` function, which is responsible for moving tokens between addresses while enforcing balance checks and applying transaction taxes. The `taxHandler.getTax` method determines the amount of tax to be deducted from each transaction. However, the contract lacks a mechanism to enforce a maximum tax limit. Consequently, the `taxHandler` contract can increase the fees over the allowed limit of 25%. The owner may take advantage of it by calling the `setTaxRate` function with a high percentage value.

```
    function _transfer(address from, address to, uint256 amount)
internal virtual {
        ...
        treasuryHandler.beforeTransferHandler(from, to, amount);

        uint256 tax = taxHandler.getTax(from, to, amount);
        uint256 taxedAmount = amount - tax;

        unchecked {
            _balances[from] = fromBalance - amount;
            // Overflow not possible: the sum of all balances is capped
by totalSupply, and the sum is preserved by
            // decrementing then incrementing.
            _balances[to] += taxedAmount;
        }
    ...
    }

    function setTaxRate(uint256 taxRate) external onlyOwner {
        _taxRate = taxRate;
    }
```

## Recommendation

The contract could embody a check for the maximum acceptable value. The team should carefully manage the private keys of the owner’s account who has the authority to set the `taxHandler` address. We strongly recommend incorporating the tax fee logic directly within the contract's codebase, rather than relying on an external contract. This approach ensures that the fee calculation logic is transparent, auditable, and securely integrated into the contract, minimizing external dependencies and potential attack vectors.

## US - Untrusted Source

| | |
|---|---|
| **Criticality** | Critical |
| **Location** | contracts/Catpurr.sol#L217,230,272 |
| **Status** | Unresolved |

### Description

The contract uses the `taxHandler` and `treasuryHandler` external contracts in order to determine the transaction's flow. The owner has the authority to change the external contracts that are being used. As a result, it may produce security issues and harm the transactions.

```
    function setTaxHandler(address taxHandlerAddress) external
onlyOwner {
        address oldTaxHandlerAddress = address(taxHandler);
        taxHandler = ITaxHandler(taxHandlerAddress);

        emit TaxHandlerChanged(oldTaxHandlerAddress,
taxHandlerAddress);
    }

    function setTreasuryHandler(address treasuryHandlerAddress)
external onlyOwner {
        address oldTreasuryHandlerAddress =
address(treasuryHandler);
        treasuryHandler = ITreasuryHandler(treasuryHandlerAddress);

        emit TreasuryHandlerChanged(oldTreasuryHandlerAddress,
treasuryHandlerAddress);
    }

    function _transfer(address from, address to, uint256 amount)
internal virtual {
        ...
        treasuryHandler.beforeTransferHandler(from, to, amount);

        uint256 tax = taxHandler.getTax(from, to, amount);
        ...

        treasuryHandler.afterTransferHandler(from, to, amount);

        ...
    }
```

## Recommendation

The contract should use only 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. As a result, consider renouncing the ownership of the `CATOURR` contract. This will prevent any future set of new external contracts.

## IEW - Inaccessible ETH Withdraw

| | |
|---|---|
| **Criticality** | Medium |
| **Location** | contracts/treasury/TreasuryHandler.sol#L59 |
| **Status** | Unresolved |

### Description

The contract is designed to facilitate the withdrawal of either ERC20 tokens or ETH to a specified treasury address based on the provided `tokenAddress`. A require statement is employed to ensure that the `tokenAddress` is not the zero address, intending to prevent erroneous transactions. However, the function also contains logic to handle ETH transfers specifically when the `tokenAddress` is the zero address. Due to the preceding `require` statement that prohibits the `tokenAddress` from being the zero address, the condition to execute the ETH transfer logic becomes unreachable. This contradiction results in a scenario where the intended functionality to allow ETH withdrawals is effectively nullified, rendering the contract unable to perform ETH transfers even though it includes code intended for that purpose.

```
   function withdraw(address tokenAddress, uint256 amount) external
onlyOwner {
        require(
            tokenAddress != address(0),
            "TreasuryHandler:withdraw:ZERO_ADDRESS: Cannot withdraw
zero address."
        );

        if (tokenAddress == address(0)) {
            // Transfer ETH to treasury
            treasury.sendValue(amount);
        } else {
            // Transfer tokens to treasury
            if (
                IERC20(tokenAddress).transfer(address(treasury),
amount)
            ) {
                emit Withdrawal(tokenAddress, amount);
            }
        }
    }
```

## Recommendation

It is recommended to revise the function's logic to correctly facilitate the withdrawal of ETH alongside ERC20 tokens. This can be achieved by removing the initial `require` statement that checks for the zero address and instead directly implementing the conditional logic to distinguish between ETH and ERC20 token withdrawals. For ETH withdrawals, a designated pseudo-address (e.g., using address(0) explicitly for ETH) can be checked without the `require` statement, allowing the function to execute ETH transfer logic when appropriate. Additionally, ensure the treasury address is capable of receiving ETH (i.e., it is an address payable). This approach will enable the contract to support both ERC20 token and ETH withdrawals as intended, enhancing its functionality and flexibility.

## IEE - Inconsistent Event Emission

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | contracts/treasury/TreasuryHandler.sol#L59 |
| **Status** | Unresolved |

### Description

The contract is designed to allow withdrawals of both ERC20 tokens and ETH to a specified treasury address. It includes a mechanism to emit a `Withdrawal` event when ERC20 tokens are successfully transferred. However, it lacks a corresponding event emission for ETH withdrawals. This inconsistency in event handling means that while token transfers are transparent and trackable through emitted events, ETH transfers occur without such transparency. This omission can lead to challenges in monitoring and verifying ETH withdrawals, affecting operational transparency of the contract.

```
    function withdraw(address tokenAddress, uint256 amount)
external onlyOwner {
        require(
            tokenAddress != address(0),
            "TreasuryHandler:withdraw:ZERO_ADDRESS: Cannot
withdraw zero address."
        );

        if (tokenAddress == address(0)) {
            // Transfer ETH to treasury
            treasury.sendValue(amount);
        } else {
            // Transfer tokens to treasury
            if (

IERC20(tokenAddress).transfer(address(treasury), amount)
            ) {
                emit Withdrawal(tokenAddress, amount);
            }
        }
    }
```

## Recommendation

It is recommended to ensure consistent event handling by emitting a `Withdrawal` event for ETH withdrawals, similar to the handling for ERC20 token withdrawals. This can be achieved by adding an emit `Withdrawal` statement within the conditional branch that emits ETH transfers. This adjustment will provide a consistent and transparent record of all withdrawals, regardless of whether they involve ERC20 tokens or ETH, enhancing the contract's transparency and making it easier to monitor the transaction history.

## PBV - Percentage Boundaries Validation

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | contracts/tax/ExponentialTaxHandler.sol#L68 |
| **Status** | Unresolved |

### Description

The contract is using the variable `_taxRate` calculations. However, this variable is used in multiplication operations and if `_taxRate` is set to a value greater than `10000`, it could lead to incorrect calculations, potentially causing unintended behavior or financial discrepancies within the contract's operations.

```
    function getTax(
        address benefactor,
        address beneficiary,
        uint256 amount
    ) external view override returns (uint256) {
        ...
        return (amount * _taxRate) / 10000;
    }
```

### Recommendation

It is recommended to ensure that the value of `_taxRate` cannot exceed `10000`. This can be achieved by adding checks whenever this variable is set.

## RSW - Redundant Storage Writes

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | contracts/tax/ExponentialTaxHandler.sol#L41,47 |
| **Status** | Unresolved |

### Description

The contract modifies the state of the following variables without checking if their current value is the same as the one given as an argument. As a result, the contract performs redundant storage writes, when the provided parameter matches the current state of the variables, leading to unnecessary gas consumption and inefficiencies in contract execution.

```
   function addExempt(address exemption) external onlyOwner {
        if (_exempted.add(exemption)) {
            emit TaxExemptionUpdated(exemption, true);
        }
    }

    function removeExempt(address exemption) external onlyOwner
{
        if (_exempted.remove(exemption)) {
            emit TaxExemptionUpdated(exemption, false);
        }
    }
```

### Recommendation

The team is advised to implement additional checks within to prevent redundant storage writes when the provided argument matches the current state of the variables. By incorporating statements to compare the new values with the existing values before proceeding with any state modification, the contract can avoid unnecessary storage operations, thereby optimizing gas usage.

## L07 - Missing Events Arithmetic

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | contracts/tax/ExponentialTaxHandler.sol#L34 |
| **Status** | Unresolved |

### Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
_taxRate = taxRate
```

### Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

## L09 - Dead Code Elimination

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | @openzeppelin/contracts/utils/Context.sol#L25 |
| **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.

## L16 - Validate Variable Setters

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | contracts/treasury/TreasuryHandler.sol#L19 |
| **Status** | Unresolved |

### Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
treasury = payable(_treasury)
```

### Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

## L18 - Multiple Pragma Directives

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | contracts/treasury/ITreasuryHandler.sol#L2<br>contracts/tax/ITaxHandler.sol#L2<br>contracts/interfaces/IERC20Burnable.sol#L2<br>contracts/Catpurr.sol#L2<br>@openzeppelin/contracts/utils/Context.sol#L4<br>@openzeppelin/contracts/token/ERC20/IERC20.sol#L4<br>@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#L4<br>@openzeppelin/contracts/access/Ownable.sol#L4 |
| **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;
pragma solidity ^0.8.17;
```

### 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/treasury/ITreasuryHandler.sol#L2<br>contracts/tax/ITaxHandler.sol#L2<br>contracts/interfaces/IERC20Burnable.sol#L2<br>contracts/Catpurr.sol#L2<br>@openzeppelin/contracts/utils/Context.sol#L4<br>@openzeppelin/contracts/token/ERC20/IERC20.sol#L4<br>@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#L4<br>@openzeppelin/contracts/access/Ownable.sol#L4<br>contracts/tax/ExponentialTaxHandler.sol#L2<br>contracts/treasury/TreasuryHandler.sol#L2<br>contracts/treasury/ITreasuryHandler.sol#L2 |
| **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 ^0.8.17;
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 |
| | | | | |
| CATPURR | Implementation | Context, IERC20, IERC20Metadata, Ownable, IERC20Burnable | | |
| | | Public | ✓ | - |
| | name | Public | | - |
| | symbol | Public | | - |
| | decimals | Public | | - |
| | totalSupply | Public | | - |
| | balanceOf | Public | | - |
| | transfer | Public | ✓ | - |
| | allowance | Public | | - |
| | approve | Public | ✓ | - |
| | transferFrom | Public | ✓ | - |
| | increaseAllowance | Public | ✓ | - |
| | decreaseAllowance | Public | ✓ | - |
| | setTaxHandler | External | ✓ | onlyOwner |
| | setTreasuryHandler | External | ✓ | onlyOwner |

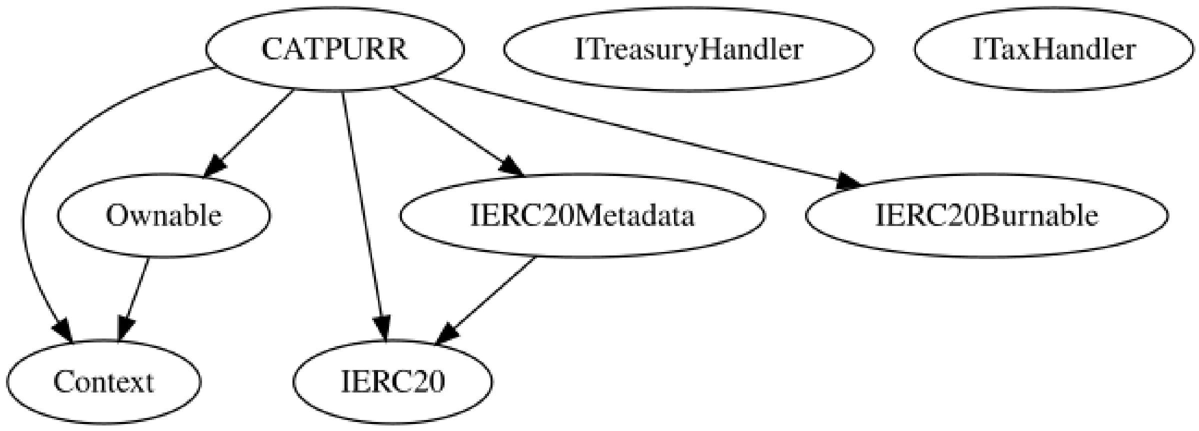
| | | | | |
|---|---|---|---|---|
| | burn | Public | ✓ | - |
| | burnFrom | Public | ✓ | - |
| | _transfer | Internal | ✓ | |
| | _mint | Internal | ✓ | |
| | _burn | Internal | ✓ | |
| | _approve | Internal | ✓ | |
| | _spendAllowance | Internal | ✓ | |
| | | | | |
| **ExponentialTaxHandler** | Implementation | ITaxHandler, Ownable | | |
| | | Public | ✓ | - |
| | setTaxRate | External | ✓ | onlyOwner |
| | getTaxRate | External | | - |
| | addExempt | External | ✓ | onlyOwner |
| | removeExempt | External | ✓ | onlyOwner |
| | isExempt | External | | - |
| | getTax | External | | - |
| | | | | |
| **TreasuryHandler** | Implementation | ITreasuryHandler, Ownable | | |
| | | Public | ✓ | - |
| | beforeTransferHandler | External | ✓ | - |
| | afterTransferHandler | External | ✓ | - |
| | setTreasury | External | ✓ | onlyOwner |
| | withdraw | External | ✓ | onlyOwner |

| | | | | |
|---|---|---|---|---|
| | | | | |
| **ITreasuryHandler** | Interface | | | |
| | beforeTransferHandler | External | ✓ | - |
| | afterTransferHandler | External | ✓ | - |
| | | | | |
| **ITaxHandler** | Interface | | | |
| | getTax | External | | - |
| | | | | |
| **IERC20Burnable** | Interface | | | |
| | burn | External | ✓ | - |
| | burnFrom | External | ✓ | - |
| | | | | |
| **Context** | Implementation | | | |
| | _msgSender | Internal | | |
| | _msgData | Internal | | |
| | _contextSuffixLength | Internal | | |
| | | | | |
| **IERC20** | Interface | | | |
| | totalSupply | External | | - |
| | balanceOf | External | | - |
| | transfer | External | ✓ | - |
| | allowance | External | | - |
| | approve | External | ✓ | - |

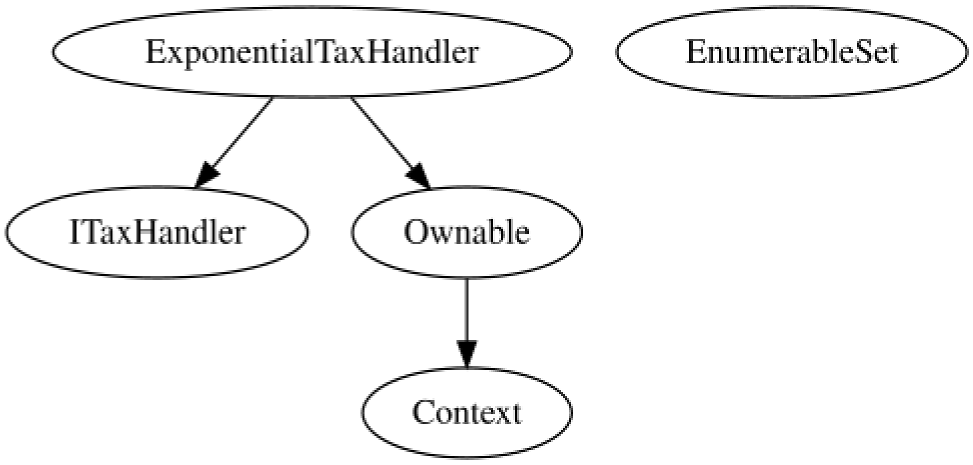
| | | | | |
|---|---|---|---|---|
| | transferFrom | External | ✓ | - |
| | | | | |
| **IERC20Metadata** | Interface | IERC20 | | |
| | name | External | | - |
| | symbol | External | | - |
| | decimals | External | | - |
| | | | | |
| **Ownable** | Implementation | Context | | |
| | | Public | ✓ | - |
| | owner | Public | | - |
| | _checkOwner | Internal | | |
| | renounceOwnership | Public | ✓ | onlyOwner |
| | transferOwnership | Public | ✓ | onlyOwner |
| | _transferOwnership | Internal | ✓ | |

# Inheritance Graph

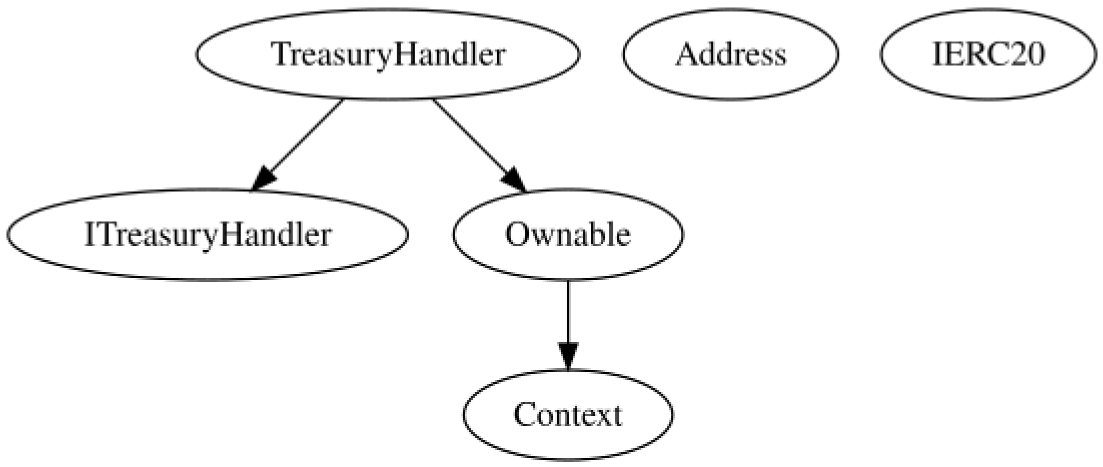
## CATPURR



## ExponentialTaxHandler

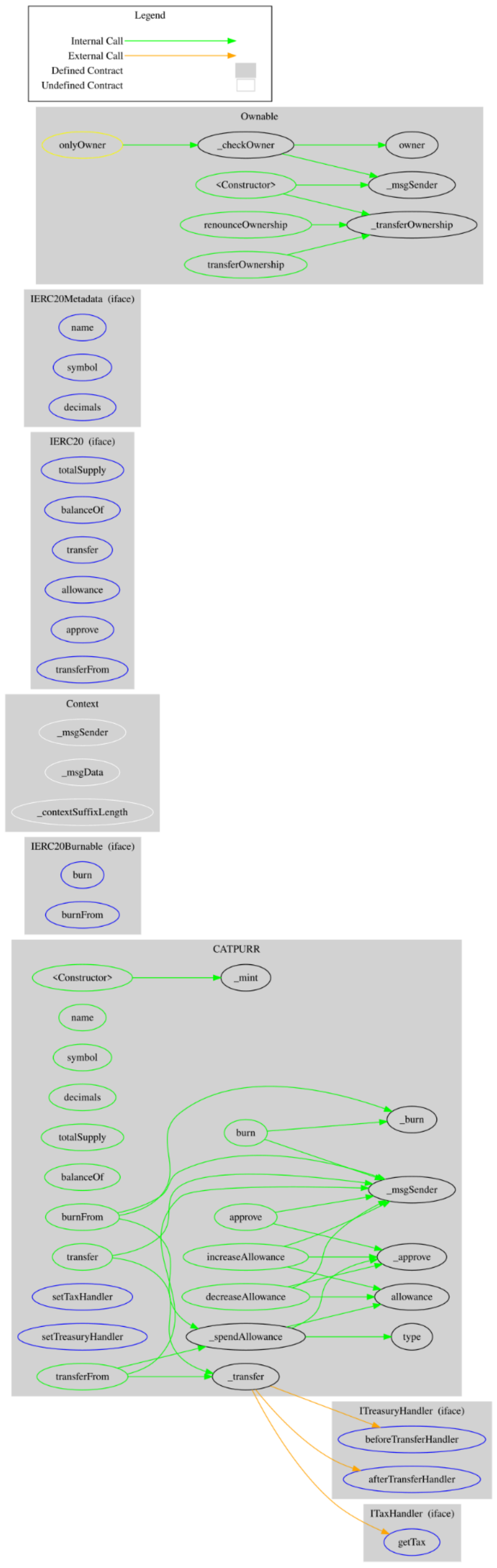


## TreasuryHandler

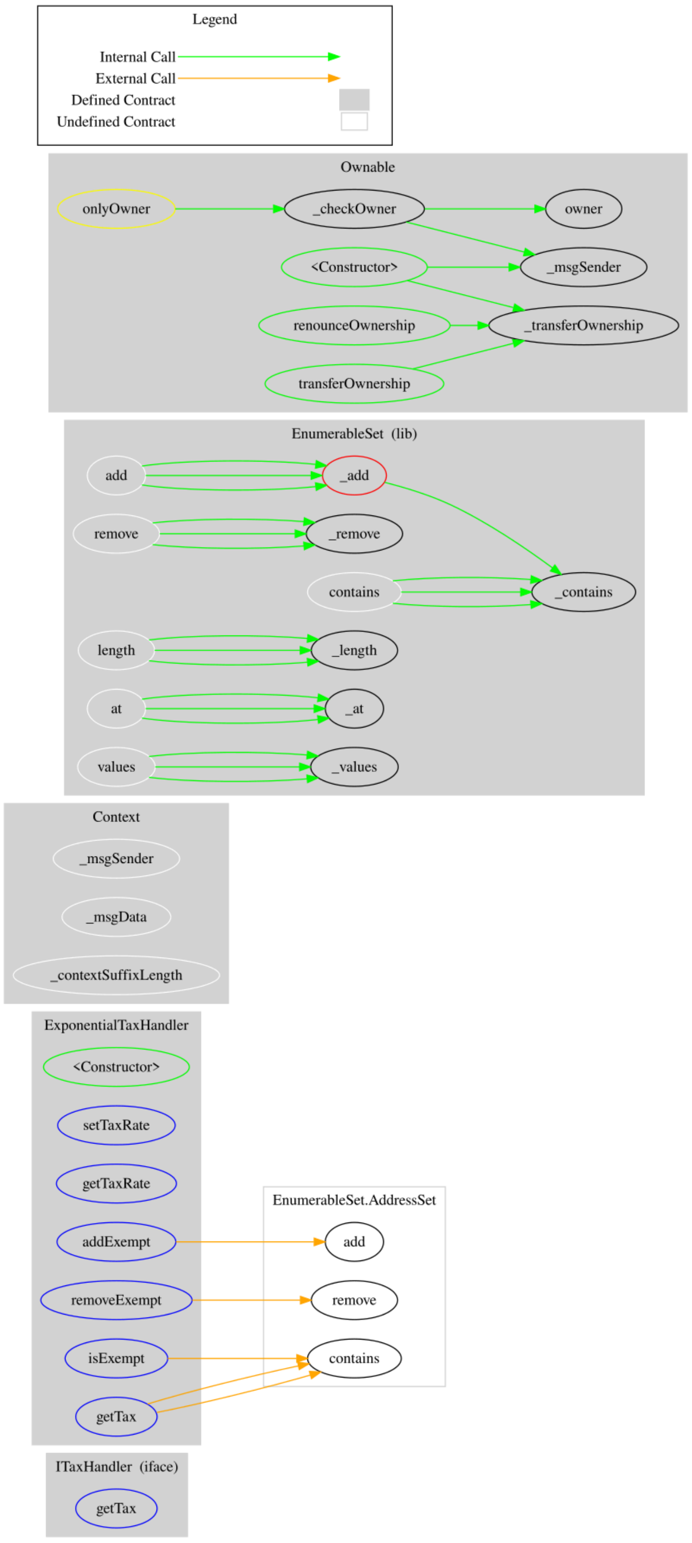


## Flow Graph

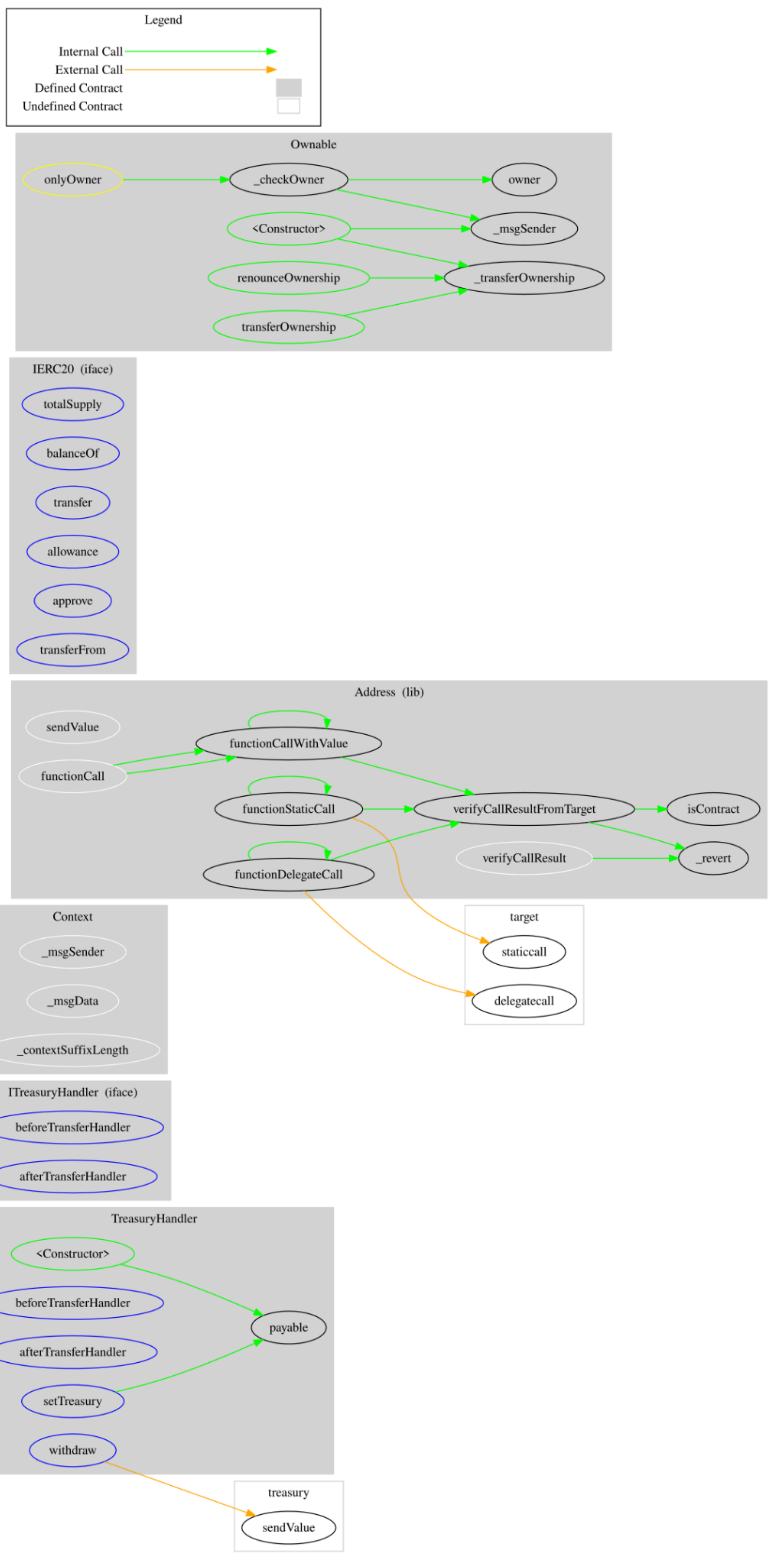
### CATPURR



## ExponentialTaxHandler



## TreasuryHandler



## Summary

Catpurr 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 stop transactions and manipulate the fees. A multi-wallet signing pattern after verifying the external source code will provide security against potential hacks.

## 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