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	Unresolved
●	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
⚪	L09	Dead Code Elimination	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

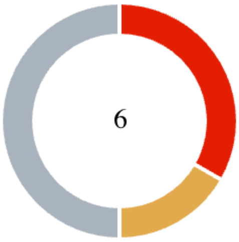
Audit Updates

Initial Audit	19 Feb 2024

Source Files

Filename	SHA256
contracts/Catpurr.sol	a81f3461e46cd9071281b9a1fc023d3b5aa1097f92db83860b14c94eed5d12aa
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	2
●	Medium	1
●	Minor / Informative	3

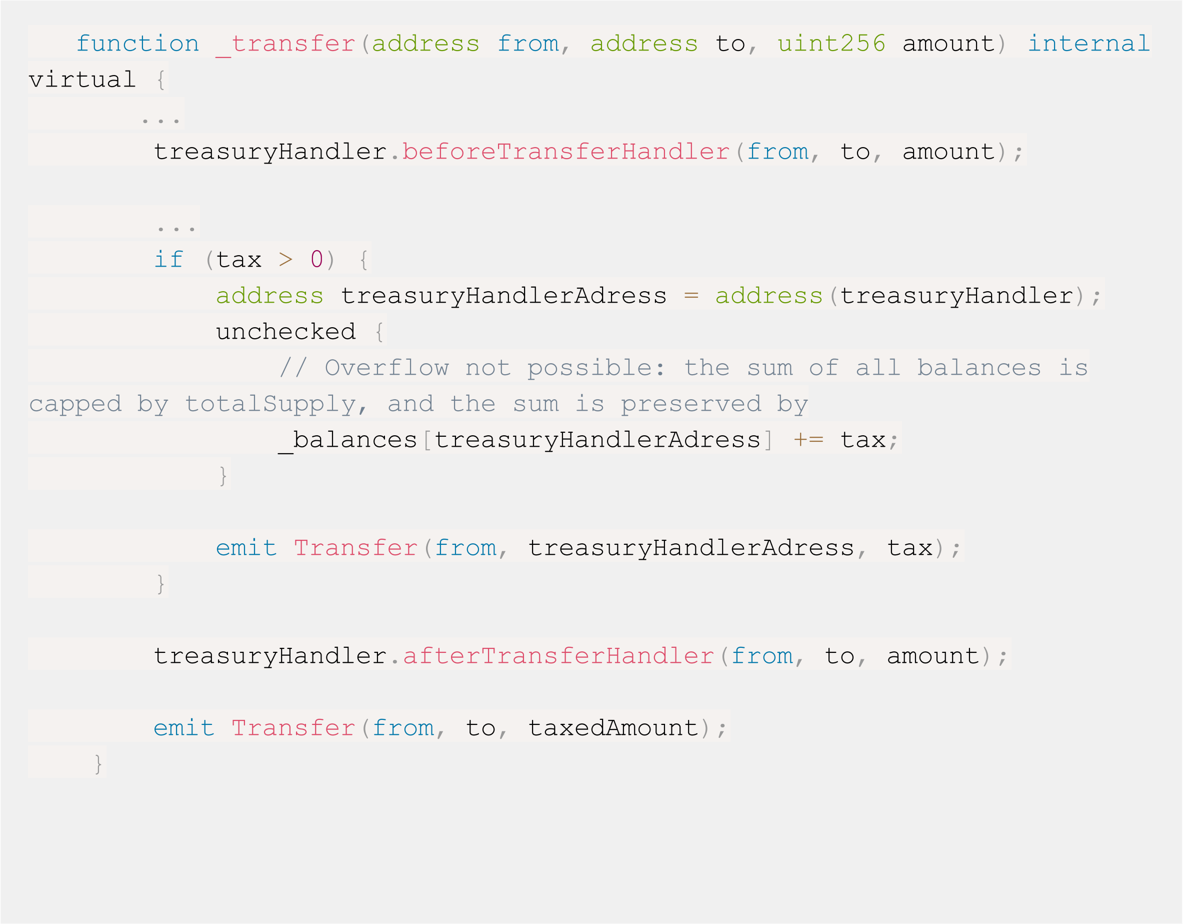
	Severity	Unresolved	Acknowledged	Resolved	Other
●	Critical	2	0	0	0
●	Medium	1	0	0	0
●	Minor / Informative	3	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	contracts/Catpurr.sol#L272
Status	Unresolved

Description

As described in the `US-Untrusted Source` finding the contract is utilizing the `treasuryHandler` external contract. This 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.

ELFM - Exceeds Fees Limit

Criticality	Medium
Location	contracts/Catpurr.sol#L281
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%.

```
   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;
        }
    ...
    }
```

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#L272
Status	Unresolved

Description

The contract uses the `taxHandler` and `treasuryHandler` external contracts in order to determine the transaction's flow. The external contracts are untrusted. As a result, it may produce security issues and harm the transactions.

```
    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 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.

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.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	contracts/treasury/ITreasuryHandler.sol#L2 contracts/tax/ITaxHandler.sol#L2 contracts/interfaces/IERC20Burnable.sol#L2 contracts/Catpurr.sol#L2 @openzeppelin/contracts/utils/Context.sol#L4 @openzeppelin/contracts/token/ERC20/IERC20.sol#L4 @openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#L4 @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 contracts/tax/ITaxHandler.sol#L2 contracts/interfaces/IERC20Burnable.sol#L2 contracts/Catpurr.sol#L2 @openzeppelin/contracts/utils/Context.sol#L4 @openzeppelin/contracts/token/ERC20/IERC20.sol#L4 @openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#L4 @openzeppelin/contracts/access/Ownable.sol#L4
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.

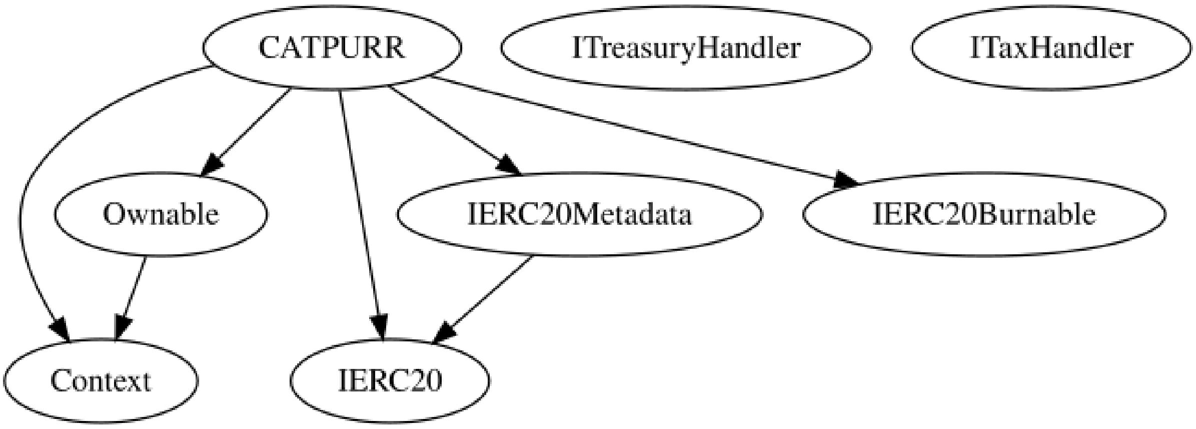
Functions Analysis

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	✓	-

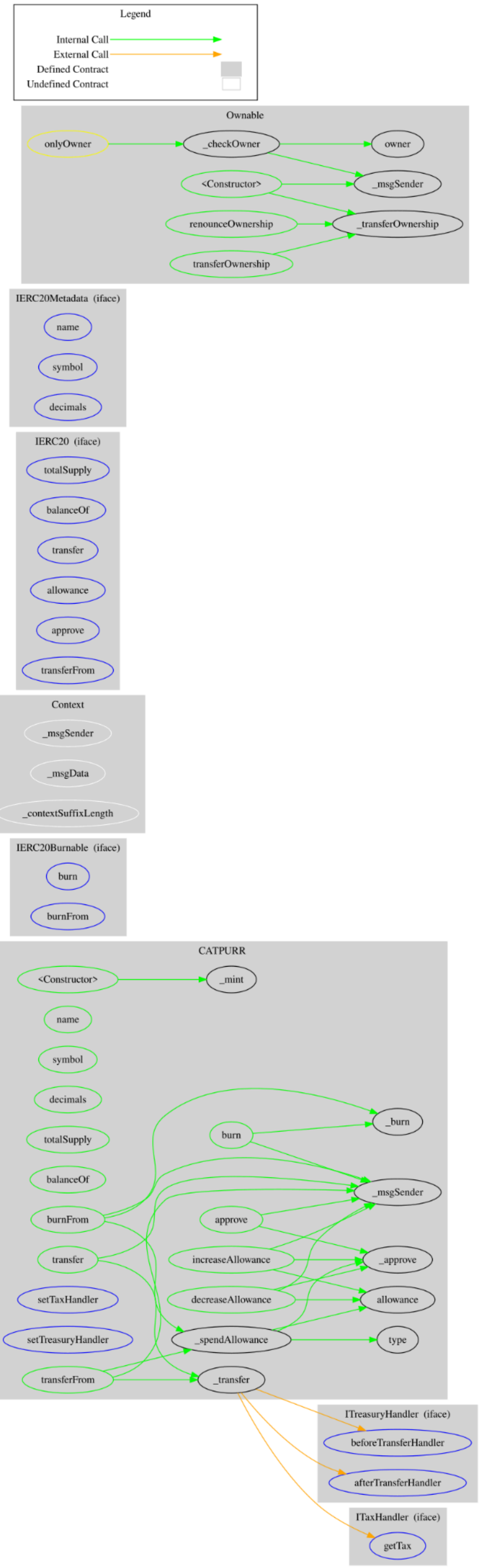
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_spendAllowance	Internal	✓	
ITreasuryHandler	Interface			
	beforeTransferHandler	External	✓	-
	afterTransferHandler	External	✓	-
ITaxHandler	Interface			
	getTax	External		-
IERC20Burnable	Interface			
	burn	External	✓	-
	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



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Ownable
onlyOwner
_checkOwner
owner
<Constructor>
_msgSender
renounceOwnership
_transferOwnership
transferOwnership
IERC20Metadata (iface)
name
symbol
decimals
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Context
_msgSender
_msgData
_contextSuffixLength
IERC20Burnable (iface)
burn
burnFrom
CATPURR
<Constructor>
_mint
name
symbol
decimals
totalSupply
burn
_burn
balanceOf
_msgSender
burnFrom
approve
transfer
increaseAllowance
_approve
setTaxHandler
decreaseAllowance
allowance
setTreasuryHandler
_spendAllowance
type
transferFrom
_transfer
ITreasuryHandler (iface)
beforeTransferHandler
afterTransferHandler
ITaxHandler (iface)
getTax

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

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