Cyberscope

Audit Report

MitaoCat

June 2023

Network ETH

Address 0x1a402bc30898eca5d3ad12842c1b7abb25344726

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
	EIS	Excessively Integer Size	Unresolved
	RFA	Redundant Fees Accumulation	Unresolved
	MEE	Missing Events Emission	Unresolved
	RFC	Redundant Fees Calculation	Unresolved
	RS	Redundant Struct	Unresolved
	IDI	Immutable Declaration Improvement	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L07	Missing Events Arithmetic	Unresolved
	L09	Dead Code Elimination	Unresolved
	L13	Divide before Multiply Operation	Unresolved
	L15	Local Scope Variable Shadowing	Unresolved

Table of Contents

Review

Contract Name	MitaoCat
Compiler Version	v0.8.20+commit.a1b79de6
Optimization	200 runs
Explorer	https://etherscan.io/address/0x1a402bc30898eca5d3ad12842c1b7abb25344726
Address	0x1a402bc30898eca5d3ad12842c1b7abb25344726
Network	ETH
Symbol	いい猫
Decimals	18
Total Supply	800,000,000,000

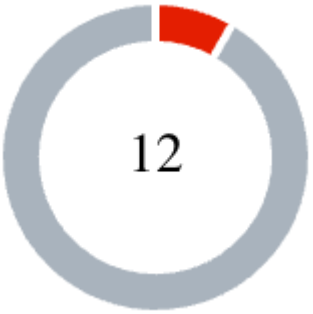
Audit Updates

Initial Audit	28 Jun 2023

Source Files

Filename	SHA256
MitaoCat.sol	ede639cc08914c97cf7496786a3870407191347e24099556672d1c61e5cb3cd1

Findings Breakdown



Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	0	0	0	0
Minor / Informative	11	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	MitaoCat.sol#L421
Status	Unresolved

Description

The transactions are initially disabled for all users excluding the authorized addresses. The owner can enable the transactions for all users. Once the transactions are enable the owner will not be able to disable them again.

```
if(!tradingActive){
    require(_isExcludedFromFees[from] || _isExcludedFromFees[to],
"Trading is not active.");
}
```

Recommendation

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. Some suggestions are:

- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

EIS - Excessively Integer Size

Criticality	Minor / Informative
Location	MitaoCat.sol#L302
Status	Unresolved

Description

The contract is using a bigger unsigned integer data type that the maximum size that is required. By using an unsigned integer data type larger than necessary, the smart contract consumes more storage space and requires additional computational resources for calculations and operations involving these variables. This can result in higher transaction costs, longer execution times, and potential scalability bottlenecks.

Since the `currentTID` cannot be greater than 3 and the max fee value is 25.

```
uint256 operationsFee;

uint256 public buyTotalFees;
uint256 public sellTotalFees;
uint256 public currentTID = 0;

function switchTaxStructure(uint256 tID) external onlyOwner {
    require(currentTID < tID && currentTID != 3, "You can only go
up to TID 3.");
    buyFees memory buyFee = _buyFees[tID];
    sellFees memory sellFee = _sellFees[tID];
    buyTotalFees = buyFee.operationsFee;
    sellTotalFees = sellFee.operationsFee;
    currentTID = tID;
}
```

Recommendation

To address the inefficiency associated with using an oversized unsigned integer data type, it is recommended to accurately determine the required size based on the range of values the variable needs to represent.

We can safely assume that all the abovementioned variables could be defined as `uint8` .

RFA - Redundant Fees Accumulation

Criticality	Minor / Informative
Location	MitaoCat.sol#L448
Status	Unresolved

Description

The contract accumulates the collected fees in a variable called `tokensForOperations`. This variable is private so it is not visible to the users. Additionally, it is accessed and reset in the swap method but it is not used in a meaningful way. As a result, it increases unnecessarily the gas consumption and decreases the code readability.

```
if (automatedMarketMakerPairs[to] && sellTotalFees > 0){
    sellFees memory sellFee = _sellFees[currentTID];
    fees = amount * sellTotalFees / 100;
    tokensForOperations += fees * sellFee.operationsFee /
sellTotalFees;
}
...
uint256 totalTokensToSwap = tokensForOperations;

if(contractBalance == 0 || totalTokensToSwap == 0) {return;}

if(contractBalance > swapTokensAtAmount * 60){
    contractBalance = swapTokensAtAmount * 60;
}

bool success;
swapTokensForEth(contractBalance);

tokensForOperations = 0;
```

Recommendation

The team is adviced to remove the state variable `tokensForOperations`.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	MitaoCat.sol#L515
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 setOperationsAddress(address _operationsAddress) external
onlyOwner {
    require(_operationsAddress != address(0), "_operationsAddress
address cannot be 0");
    operationsAddress = _operationsAddress;
}
```

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.

RFC - Redundant Fees Calculation

Criticality	Minor / Informative
Location	MitaoCat.sol#L445
Status	Unresolved

Description

The `sellTotalFees` are always equal to the `sellFees.operationsFee`. As a result, the expression `sellFee.operationsFee / sellTotalFees` will always equal 1. The same applies to the buy variable.

```
sellFees memory sellFee = _sellFees[currentTID];
fees = amount * sellTotalFees / 100;
tokensForOperations += fees * sellFee.operationsFee /
sellTotalFees;
...
buyFees memory buyFee = _buyFees[currentTID];
fees = amount * buyTotalFees / 100;
tokensForOperations += fees * buyFee.operationsFee / buyTotalFees;
```

Recommendation

The team is advised to remove the division of the fee since it always produces multiplication with the number one.

RS - Redundant Struct

Criticality	Minor / Informative
Location	MitaoCat.sol#L302
Status	Unresolved

Description

The contract uses two structures, one for buys and one for sales. These structs contain only one `uint256` variable. Since it contains only one property then the struct is redundant.

```
struct sellFees {
    uint256 operationsFee;
}

struct buyFees {
    uint256 operationsFee;
}
```

Recommendation

The team is advised to remove the structs and move the internal property as a contract state variable.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	MitaoCat.sol#L355,357,362,369,370
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.

```
dexRouter
lpPair
swapTokensAtAmount
XiaOXuai
CEXMitao
```

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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	MitaoCat.sol#L278,298,302,313,314,511
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.

```
on WETH() external pure returns (address);


 sellFees {
        uint256 operationsFee;
...

 buyFees {
        uint256 operationsFee;
    }


...
```

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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	MitaoCat.sol#L404
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.

```
tID;
    }
```

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	MitaoCat.sol#L218
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.

```
on _burn(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: burn from the zero
address");
        uint256 accountBalance = _balances[account];
        require(accountBalance >= amount, "ERC20: burn amount
exceeds balance");
        unchecked {
            _balances[account] = accountBalance - amount;
            // Overflow not possible: amount <= accountBalance <=
totalSupply.
            _balances[address(0xdead)] += amount;
        }

        emit Transfer(account, address(0xdead), amount);
    }
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	MitaoCat.sol#L444,449,450
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
t * buyTotalFees / 100;

rations += fees * buyFee.operationsFee / buyTotalFees;
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	MitaoCat.sol#L360
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.

```
lSupply = 800 * 1e9 * 1e18;
```

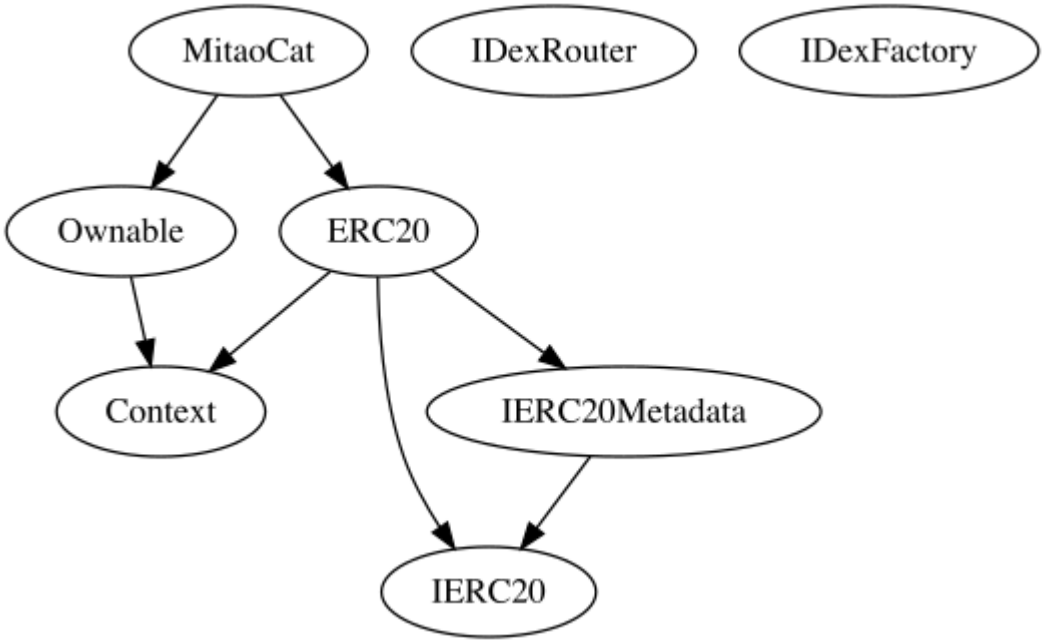
Recommendation

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.

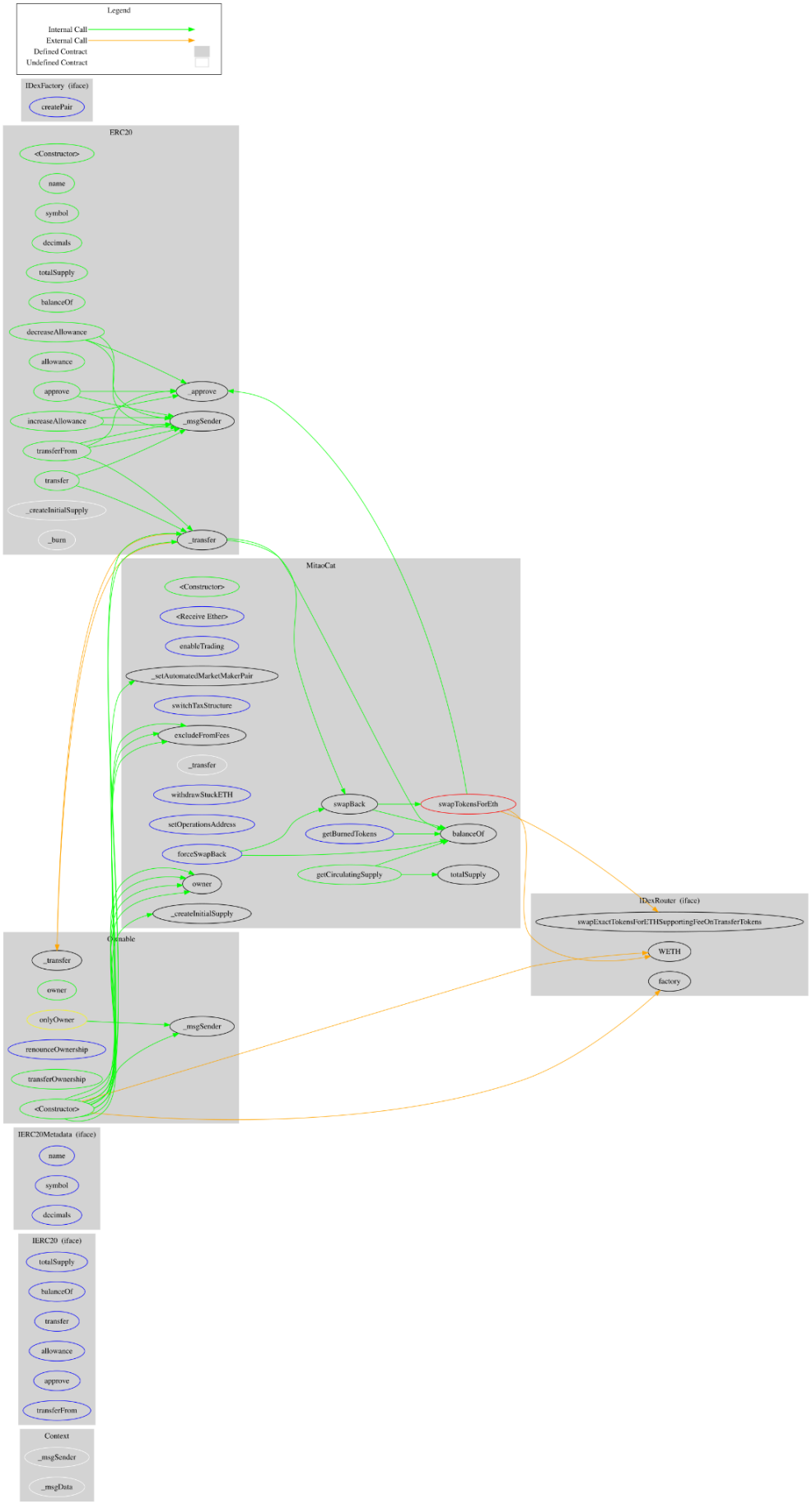
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
MitaoCat	Implementation	ERC20, Ownable		
		Public	✓	ERC20
		External	Payable	-
	enableTrading	External	✓	onlyOwner
	_setAutomatedMarketMakerPair	Private	✓	
	switchTaxStructure	External	✓	onlyOwner
	excludeFromFees	Public	✓	onlyOwner
	_transfer	Internal	✓	
	swapTokensForEth	Private	✓	
	getBurnedTokens	External		-
	getCirculatingSupply	Public		-
	swapBack	Private	✓	
	withdrawStuckETH	External	✓	onlyOwner
	setOperationsAddress	External	✓	onlyOwner
	forceSwapBack	External	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IDexFactory (iface)
createPair
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
decreaseAllowance
allowance
approve
increaseAllowance
transferFrom
transfer
_createInitialSupply
_burn
_approve
_msgSender
_transfer
MitaoCat
<Constructor>
<Receive Ether>
enableTrading
_setAutomatedMarketMakerPair
switchTaxStructure
excludeFromFees
_transfer
withdrawStuckETH
setOperationsAddress
forceSwapBack
owner
_createInitialSupply
swapBack
swapTokensForEth
getBurnedTokens
balanceOf
getCirculatingSupply
totalSupply
IDexRouter (iface)
swapExactTokensForETHSupportingFeeOnTransferTokens
WETH
factory
Ownable
_transfer
owner
onlyOwner
renounceOwnership
transferOwnership
<Constructor>
_msgSender
IERC20Metadata (iface)
name
symbol
decimals
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Context
_msgSender
_msgData

Summary

MitaoCat 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. A multi-wallet signing pattern will provide security against potential hacks. There is also a limit of max 25% fees.

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