Cyberscope

Audit Report

MicroPets

October 2023

Network	BSC
Address	0x12673b5b0f249512b781f13e355c345650ab866b
Audited by	© cyberscope

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	OTUT	Transfers User's Tokens	Acknowledged
●	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
	RV	Redundant Variable	Acknowledged
	PTAI	Potential Transfer Amount Inconsistency	Acknowledged
	MTC	Misleading Tax Collection	Unresolved
	RSML	Redundant SafeMath Library	Unresolved
	L02	State Variables could be Declared Constant	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L07	Missing Events Arithmetic	Unresolved
	L09	Dead Code Elimination	Unresolved
	L17	Usage of Solidity Assembly	Unresolved
	L19	Stable Compiler Version	Unresolved
	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Analysis **1**
Diagnostics **2**
Table of Contents **3**
Review **5**
Audit Updates 5
Source Files 5
Findings Breakdown **6**
ST - Stops Transactions 7
Description 7
Recommendation 7
OTUT - Transfers User's Tokens 8
Description 8
Recommendation 9
Team Update 9
RV - Redundant Variable 10
Description 10
Recommendation 10
Team Update 11
PTAI - Potential Transfer Amount Inconsistency 12
Description 12
Recommendation 13
Team Update 13
MTC - Misleading Tax Collection 14
Description 14
Recommendation 15
RSML - Redundant SafeMath Library 16
Description 16
Recommendation 16
L02 - State Variables could be Declared Constant 17
Description 17
Recommendation 17
L04 - Conformance to Solidity Naming Conventions 18
Description 18
Recommendation 19
L07 - Missing Events Arithmetic 20
Description 20
Recommendation 20
L09 - Dead Code Elimination 21
Description 21

Recommendation 21
L17 - Usage of Solidity Assembly 22
Description 22
Recommendation 22
L19 - Stable Compiler Version 23
Description 23
Recommendation 23
L20 - Succeeded Transfer Check 24
Description 24
Recommendation 24
Functions Analysis **25**
Inheritance Graph **28**
Flow Graph **29**
Summary **30**
Disclaimer **31**
About Cyberscope **32**

Review

Contract Name	MicroPets
Compiler Version	v0.8.16+commit.07a7930e
Optimization	200 runs
Explorer	https://bscscan.com/address/0x12673b5b0f249512b781f13e355c345650ab866b
Address	0x12673b5b0f249512b781f13e355c345650ab866b
Network	BSC
Decimals	18

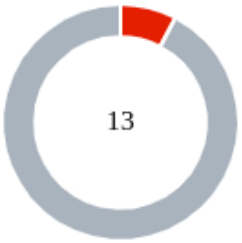
Audit Updates

Initial Audit	30 Oct 2023 https://github.com/cyberscope-io/audits/blob/main/dc-pets/v1/audit.pdf
Corrected Phase 2	07 Nov 2023

Source Files

Filename	SHA256
Utils.sol	a81996b7539309494b258d006efc063124a085c5e27e182edaabe4d1358eadb5
Uniswap.sol	375dc68512149365bf130dc289ed683898eed5a7ba9547abd31b7c897e3d8bc8
Micropets.sol	b599a83e17ac9dd6059220729c1f7f62562054bcc6ae5bc767836a80358ec099

Findings Breakdown



Critical	1
Medium	0
Minor / Informative	12

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	0	0	0	0
Minor / Informative	9	3	0	0

ST - Stops Transactions

Criticality	Critical
Location	Micropets.sol#L282
Status	Unresolved

Description

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

```
require(enableUniSwap, "Uniswap not enabled yet");
```

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.

OTUT - Transfers User's Tokens

Criticality	Minor / Informative
Location	Micropets.sol#L674
Status	Acknowledged

Description

The contract allows users to migrate their token balances from an external address (`0xA77346760341460B42C230ca6D21d4c8E743Fa9c`) to the current contract address using the `migrate` function. However, users have the authority to transfer their balance of the token from another address, which remains out of the audit scope, to the current contract address. As a result a large amount of tokens can be transferred to the current address, leading to inconsistencies in the token balances of this address. The transferred amount can enable users to migrate tokens from the previous address, which might not have any value, to this address where the tokens might have value. This could artificially inflate the value of the tokens in the current contract.

```
function migrate() external {
        require(migrationRunning, "Migration over");

        uint256 V1Balance =
IERC20(0xA77346760341460B42C230ca6D21d4c8E743Fa9c)
            .balanceOf(_msgSender());

        require(V1Balance > 0, "Invalid migration");

        safeTransferFrom(
            0xA77346760341460B42C230ca6D21d4c8E743Fa9c,
            _msgSender(),
            address(this),
            V1Balance
        );

        uint256 newTokenAmount = V1Balance.div(migrationRate);
        _transferStandard(migrationVault, _msgSender(),
newTokenAmount);
    }
```

Recommendation

It is recommended to implement a more streamlined method for token migration. The contract should be designed to handle the receipt of tokens in a manner that prevents any inconsistencies. Specifically, each migration could be limited to a fixed percentage of each holder's balance, ensuring that no user can migrate an excessive amount at once. This approach will ensure that migrations are conducted in a controlled manner, reducing the risk of sudden, large-scale migrations that could disrupt the token's ecosystem.

Team Update

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

Since both the old contract and the new one have fixed supplies, there is no need to limit the token amounts than can be migrated, should the migration be enabled again, the migration rates will have to be updated as well, and people will be informed.

RV - Redundant Variable

Criticality	Minor / Informative
Location	Micropets.sol#L72,553,591
Status	Acknowledged

Description

The contract contains the `autoSwapTier` variable within the `Configs` struct. This variable can is set to `500` through the `setShares` functions. However, the `autoSwapTier` variable is not utilized in any of the contract's functionalities. This makes the variable redundant and can lead to higher gas consumption.

```
    struct Configs {
        ...
        uint24 autoSwapTier;
    }

    function setShares(
        ...
        uint24 autoSwapTier
    ) external onlyOwner {
        _setShares(
            ...
            autoSwapTier
        );
    }

    function setAutoSwapTier(uint24 autoSwapTier) external
onlyOwner {
        tokenConfigs.autoSwapTier = autoSwapTier;
    }
```

Recommendation

It is recommended to remove the `autoSwapTier` variable from the `Configs` struct. Keeping unused variables not only increases the gas cost for contract deployment but also makes the codebase less maintainable and more prone to errors in the future. Cleaning up

such redundant elements will streamline the contract, making it more efficient and easier to understand.

Team Update

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

The variable `autoSwapTier` *will be needed for uniswap V3 later on*

PTAI - Potential Transfer Amount Inconsistency

Criticality	Minor / Informative
Location	Micropets.sol#L674
Status	Acknowledged

Description

The `transfer()` and `transferFrom()` functions are used to transfer a specified amount of tokens to an address. The fee or tax is an amount that is charged to the sender of an ERC20 token when tokens are transferred to another address. According to the specification, the transferred amount could potentially be less than the expected amount. This may produce inconsistency between the expected and the actual behavior.

The following example depicts the diversion between the expected and actual amount.

Tax	Amount	Expected	Actual
No Tax	100	100	100
10% Tax	100	100	90

```
   function migrate() external {
        require(migrationRunning, "Migration over");

        uint256 V1Balance =
IERC20(0xA77346760341460B42C230ca6D21d4c8E743Fa9c)
            .balanceOf(_msgSender());

        require(V1Balance > 0, "Invalid migration");

        safeTransferFrom(
            0xA77346760341460B42C230ca6D21d4c8E743Fa9c,
            _msgSender(),
            address(this),
            V1Balance
        );

        uint256 newTokenAmount = V1Balance.div(migrationRate);
        _transferStandard(migrationVault, _msgSender(), newTokenAmount);
    }
...
_transferStandard(migrationVault, _msgSender(), amount);
```

Recommendation

The team is advised to take into consideration the actual amount that has been transferred instead of the expected.

It is important to note that an ERC20 transfer tax is not a standard feature of the ERC20 specification, and it is not universally implemented by all ERC20 contracts. Therefore, the contract could produce the actual amount by calculating the difference between the transfer call.

```
 Actual Transferred Amount = Balance After Transfer - Balance
Before Transfer
```

Team Update

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

The new contract is excluded from taxes with the old static contract

MTC - Misleading Tax Collection

Criticality	Minor / Informative
Location	Micropets.sol#L389
Status	Unresolved

Description

The contract contains a tax mechanism where a portion of transactions is allocated as tax and is intended to be collected and stored in the `_taxCollected` variable. However, the `swapTokensForETH` function within `swapAndLiquify` uses a conditional check to determine whether to swap the entire `tokenAmount` or just the `maxSwap` amount. If the `maxSwap` is used, the actual amount of tokens swapped for ETH will be less. Despite this, the `_taxCollected` variable is incremented by the full `tokensToSwap` amount, not accounting for the potential reduction in swapped tokens. Consequently, `_taxCollected` will inaccurately represent a higher value than the actual taxes collected, leading to a misleading state within the contract's tax accounting logic.

```
  function swapAndLiquify(uint256 tokensToSwap) internal lockForSwap {
        uint256 rewardShare = tokensToSwap
            .mul(tokenConfigs.tokenShareReserve)
            .div(100);
        _transferStandard(address(this), tokenReserveAddress,
rewardShare);
        swapTokensForETH(tokensToSwap.sub(rewardShare));
        _taxCollected = _taxCollected.add(tokensToSwap);
    }

   function swapTokensForETH(uint256 tokenAmount) internal {
        ...
        uint256 maxSwap = _balances[pair].div(100);


uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
            tokenAmount > maxSwap ? maxSwap : tokenAmount,
            ...
        );
    }
```

Recommendation

It is recommended to refactor the calculation of the `_taxCollected` variable to accurately reflect the actual amount of tax collected. The `_taxCollected` should be incremented only by the amount of tax that has been successfully transferred to the `tokenReserveAddress` plus any tokens that have been actually swapped for ETH.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	Micropets.sol
Status	Unresolved

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert to underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases gas consumption unnecessarily.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	Micropets.sol#L47
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.

```
ISwapRouter public swapRouter
```

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	Micropets.sol#L491,500,508,519,527,538,601,607,612,617
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.

```
address _addr
address payable _marketingAddress
address payable _developmentAddress
address payable _vaultAddress
address payable _coinStakingAddress
address _tokenReserveAddress
uint256 _minimumTokensBeforeSwap
uint256 _minimumETHToTransfer
bool _enabled
```

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	Micropets.sol#L603,609,626
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.

```
minimumTokensBeforeSwap = _minimumTokensBeforeSwap
minimumETHToTransfer = _minimumETHToTransfer
migrationRate = newRate
```

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	Micropets.sol#L319
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 safeTransferToken(address token, address to, uint
value) internal {
        (bool success, bytes memory data) = token.call(
            abi.encodeWithSelector(0xa9059cbb, to, value)
        );
        require(
            success && (data.length == 0 || abi.decode(data,
(bool))),
            "Failed to send token"
        );
    }
```

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	Micropets.sol#L493
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(_addr)
        }
```

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	Micropets.sol#L17
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 ^0.8.16;
```

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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	Micropets.sol#L712
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
V1.transfer(receiver, V1.balanceOf(address(this)))
```

Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

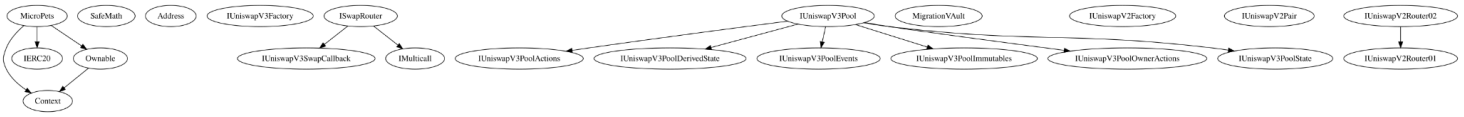
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
MicroPets	Implementation	Context, IERC20, Ownable		
	initialize	Public	✓	-
	name	External		-
	symbol	External		-
	decimals	External		-
	totalSupply	External		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
	increaseAllowance	External	✓	-
	decreaseAllowance	External	✓	-
	_approve	Private	✓	
	_transfer	Private	✓	
	handleTaxAutomation	Internal	✓	
	safeTransferETH	Internal	✓	
	safeTransferToken	Internal	✓	
	safeTransferFrom	Internal	✓	

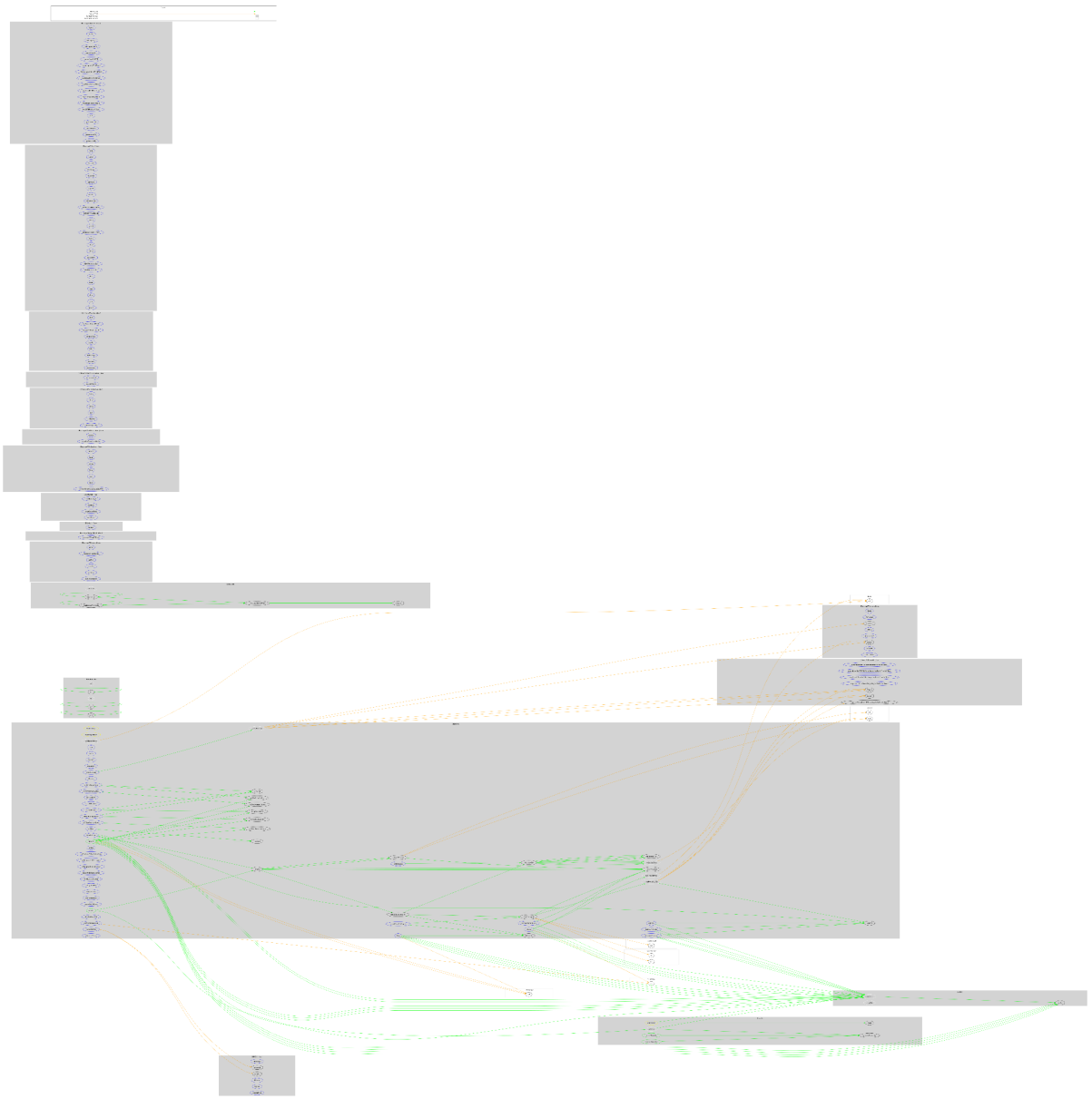
	manualSwapAndLiquify	External	✓	onlyOwner
	swapTokensForETH	Internal	✓	
	swapAndLiquify	Internal	✓	lockForSwap
	_calcuclateShare	Internal		
	_distributeTax	Internal	✓	lockForSplitShare
	distributeTax	External	✓	onlyOwner
	_transferStandard	Internal	✓	
	_transferWithTax	Internal	✓	
	includeInFee	External	✓	onlyOwner
	excludeFromFee	External	✓	onlyOwner
	_setMarketingAddress	Internal	✓	
	_setDevelopmentAddress	Internal	✓	
	_setLpVaultAddress	Internal	✓	
	_setCoinStakingAddress	Internal	✓	
	_setTokenReserveAddress	Internal	✓	
	isContract	Internal		
	setMarketingAddress	External	✓	onlyOwner
	setDevelopmentAddress	External	✓	onlyOwner
	setLpVaultAddress	External	✓	onlyOwner
	setCoinStakingAddress	External	✓	onlyOwner
	setTokenReserveAddress	External	✓	onlyOwner
	_setShares	Internal	✓	
	setShares	External	✓	onlyOwner
	getTax	External		-

	setMinimumTokensBeforeSwap	External	✓	onlyOwner
	setMinimumETHToTransfer	External	✓	onlyOwner
	setSwapAndLiquifyEnabled	External	✓	onlyOwner
	setAutoSplitSharesEnables	External	✓	onlyOwner
	setMigrationRunning	External	✓	onlyOwner
	setMigrationRate	External	✓	onlyOwner
	enableUniswap	External	✓	onlyOwner
	addPoolAddress	External	✓	onlyOwner
	removePoolAddress	External	✓	onlyOwner
	_setupExchange	Internal	✓	
	setupExchange	External	✓	onlyOwner
	totalTaxCollected	External		onlyOwner
	burn	External	✓	-
	getMigrationAmount	External		-
	migrate	External	✓	-
	retrieveOldPets	External	✓	onlyOwner
	retrieveMigrationPets	External	✓	onlyOwner
		External	Payable	-

Inheritance Graph



Flow Graph



Summary

MicroPets 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 transfer the user's tokens. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats. 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