Cyberscope

Audit Report

MicroPets

April 2024

Network	ETH
Address	0x6e65E374eA3cf0F5650F2BA9a7D8E5A1A372b37C
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
	RSML	Redundant SafeMath Library	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

Table of Contents

Review

Contract Name	MicroPets
Compiler Version	v0.8.25+commit.b61c2a91
Optimization	200 runs
Explorer	https://etherscan.io/address/0x6e65e374ea3cf0f5650f2ba9a7d8e5a1a372b37c
Address	0x6e65E374eA3cf0F5650F2BA9a7D8E5A1A372b37C
Network	ETH
Decimals	18
Badge Eligibility	Yes

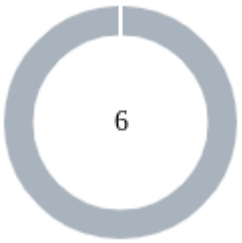
Audit Updates

Initial Audit	30 Oct 2023 https://github.com/cyberscope-io/audits/blob/main/dc-pets/v1/audit.pdf
Corrected Phase 2	07 Nov 2023 https://github.com/cyberscope-io/audits/blob/main/dc-pets/v2/audit.pdf
Corrected Phase 3	23 Nov 2023
Corrected Phase 4	20 Apr 2024

Source Files

Filename	SHA256
Utils.sol	e2d9fe84bcd373e42137f7ef35292a4807a7a568bc6a1a9d1113cd9f4525d1e9
Uniswap.sol	2737aa98fb6dbfdf508b1d4c7cbc92413dc4cdd266ceb8a1b22ba9685e1b0fd8
Micropets.sol	1be04b36bc685df179e66e

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	6

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	6	0	0	0

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 in cases where the explanatory error message is not used.

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

Recommendation

The team is advised to remove the SafeMath library in cases where the revert error message is not used. 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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	Micropets.sol#L480,489,501,516,524,539,603,609,614,619
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#L605,611
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
```

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#L302,312
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#L482
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.

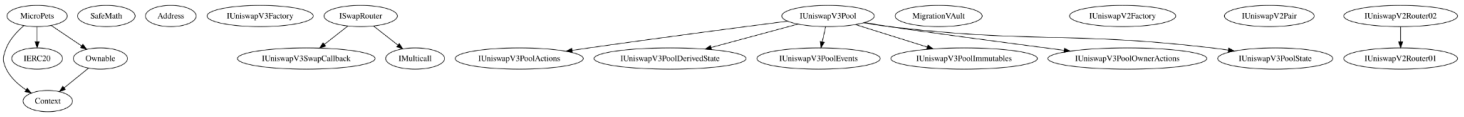
Functions Analysis

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	✓	

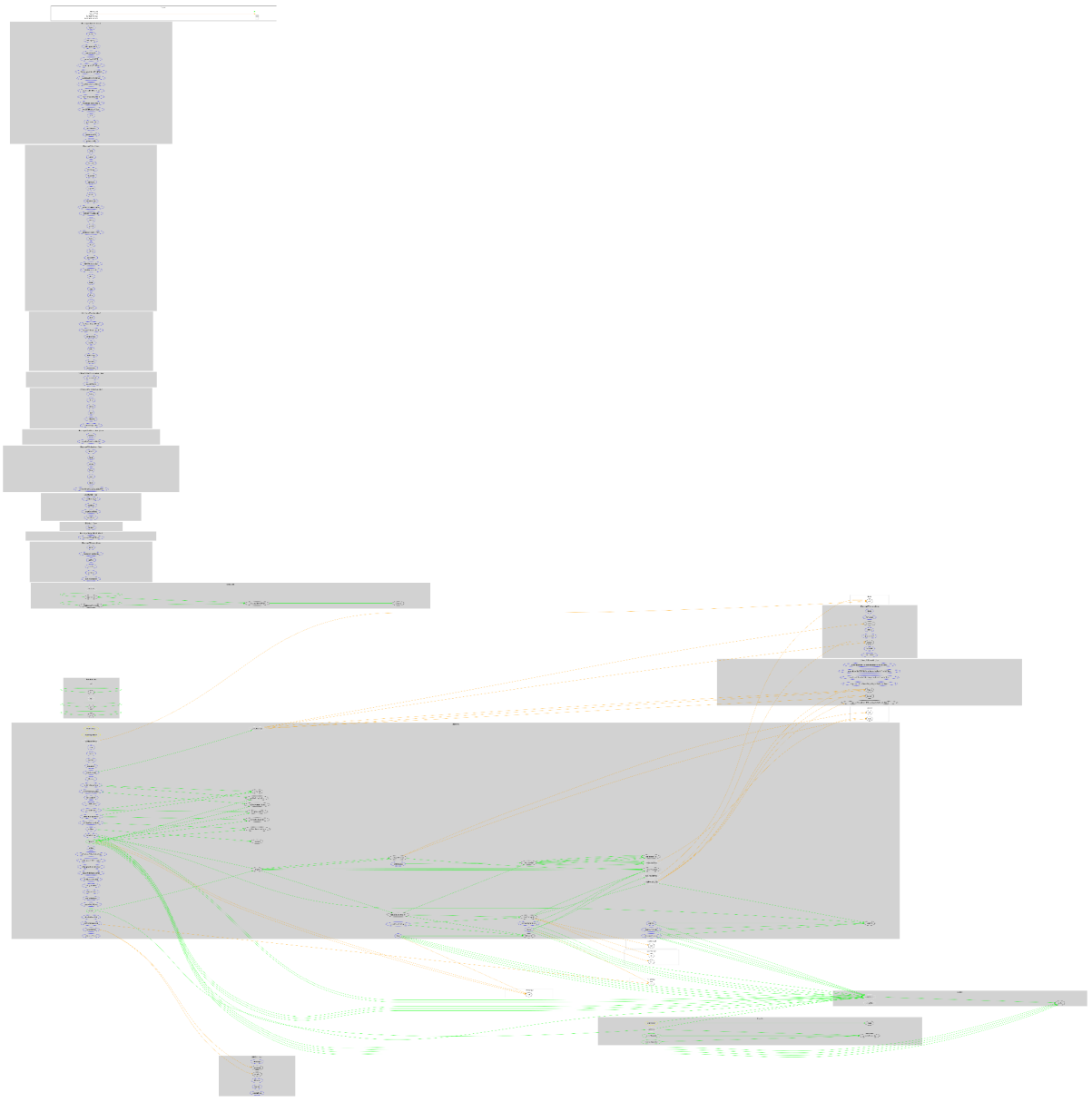
	safeTransferFrom	Internal	✓	
	manualSwapAndLiquify	External	✓	onlyOwner
	swapTokensForETH	Internal	✓	
	swapAndLiquify	Internal	✓	lockForSwap
	_calcuclateShare	Internal		
	_distributeTax	Internal	✓	lockForSplitShare
	distributeTax	External	✓	onlyOwner
	_transferStandard	Internal	✓	
	_transferWithTax	Internal	✓	
	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
	addPoolAddress	External	✓	onlyOwner
	removePoolAddress	External	✓	onlyOwner
	_setupExchange	Internal	✓	
	setupExchange	External	✓	onlyOwner
	totalTaxCollected	External		onlyOwner
	burn	External	✓	-
		External	Payable	-

Inheritance Graph



Flow Graph



Summary

MicroPets contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. 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% fee.

Corrected Phase 4, 20 Apr 2024

At the time of the audit report, the contract with address 0x6e65e374ea3cf0f5650f2ba9a7d8e5a1a372b37c is pointed out by the following proxy address: 0x2466858ab5edAd0BB597FE9f008F568B00d25Fe3.

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