Cyberscope

Audit Report

Buddha

December 2023

Network ETH

Address 0xDeFB0B264032e4e128b00D02b3FD0aA00331237b

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
●	PAV	Pair Address Validation	Unresolved
●	RCS	Redundant Conditional Statement	Unresolved
●	RAO	Redundant Addition Operation	Unresolved
●	RVD	Redundant Variable Declaration	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L13	Divide before Multiply Operation	Unresolved

Table of Contents

Review

Contract Name	Buddha
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	200 runs
Explorer	https://etherscan.io/address/0xdefb0b264032e4e128b00d02b3fd0aa00331237b
Address	0xdefb0b264032e4e128b00d02b3fd0aa00331237b
Network	ETH
Symbol	Buddha
Decimals	18
Total Supply	1,000,000,000,000

Audit Updates

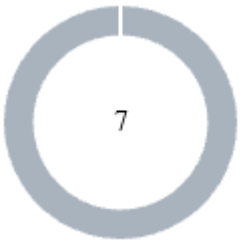
Initial Audit	21 Dec 2023

Source Files

Filename	SHA256
Token.sol	46d1b66c78e3c5c53d80dd9d85225ad0d6ff9a7d58fc3f896c68b9f4c6744924
Ownable2Step.sol	3e3bdb084bc14ade54e8259e710287956a7dbf2b2b4ad1e4cd8899d2293c7241

Ownable.sol	33422e7771fefe5fbfe8934837515097119d82a50eda0e49b38e4d6a64a1c25d
Initializable.sol	b05c26d897c4178cbdb35ad113527e463e1bdeae5764869318a54f93c8b98a94
IUniswapV2Router02.sol	a2900701961cb0b6152fc073856b972564f7c798797a4a044e83d2ab8f0e8d38
IUniswapV2Router01.sol	0439ffe0fd4a5e1f4e22d71ddbda76d63d61679947d158cba4ee0a1da60cf663
IUniswapV2Pair.sol	29c75e69ce173ff8b498584700fef76bc81498c1d98120e2877a1439f0c31b5a
IUniswapV2Factory.sol	51d056199e3f5e41cb1a9f11ce581aa3e190cc982db5771ffeef8d8d1f962a0d
IERC20Metadata.sol	b10e2f8bcc3ed53a5d9a82a29b1ad3209225331bb4de4a0459862a762cf83a1a
IERC20.sol	7ebde70853ccafcf1876900dad458f46eb9444d591d39bfc58e952e2582f5587
ERC20Burnable.sol	480b22ce348050fdb85a693e38ed6b4767a94e4776fc6806d6808a0ec171177e
ERC20.sol	f70c6ae5f2dda91a37e17cfcbec390cc59515ed0d34e316f036f5431b5c0a3f2
Context.sol	1458c260d010a08e4c20a4a517882259a23a4baa0b5bd9add9fb6d6a1549814a

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	7

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	7	0	0	0

PAV - Pair Address Validation

Criticality	Minor / Informative
Location	Token.sol#L199
Status	Unresolved

Description

The `setAMMPair` function allows any user to set any arbitrary value without validation to the `AMMPairs` mapping, which is supposed to hold Uniswap pair addresses. This lack of validation can lead to unintended behavior, including the potential disruption of the contract's intended functionality.

```
function setAMMPair(address pair, bool isPair) external onlyOwner {
    require(pair != pairV2, "DefaultRouter: Cannot remove initial pair
from list");

    _setAMMPair(pair, isPair);
}

function _setAMMPair(address pair, bool isPair) private {
    AMMPairs[pair] = isPair;

    if (isPair) {
    }

    emit AMMPairsUpdated(pair, isPair);
}
```

Recommendation

To mitigate the risks associated with the absence of address validation in the pair address argument, it is recommended to implement comprehensive address validation mechanisms. A recommended approach could be to verify pair existence in the decentralized application. Prior to interacting with the pair address contract, perform checks to verify the existence and validity of the contract at the provided address. This can be achieved by querying the provider's contract or utilizing external libraries that provide contract verification services.

RCS - Redundant Conditional Statement

Criticality	Minor / Informative
Location	Token.sol#L165
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The conditional statement `if (false || _buddhamarketingPending > 0)` in the contract is redundant. The `_buddhamarketingPending > 0` condition has already been checked earlier within the transfer function, making the code segment unnecessary and serving no purpose.

```
if (false || _buddhamarketingPending > 0) {
    ...
}
```

Additionally, the `_setAMMPair` function contains a control flow block that executes no code. As a result, the code segment is redundant.

```
if (isPair) {
}
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

RAO - Redundant Addition Operation

Criticality	Minor / Informative
Location	Token.sol#L99,166
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The contract includes code segments with redundant addition operations with zero. These additions are unnecessary, as adding any number to zero yields the same result. This redundancy does not contribute to the logic and may create confusion or code maintenance challenges.

```
function getAllPending() public view returns (uint256) {
    return 0 + _buddhamarketingPending;
}

uint256 token2Swap = 0 + _buddhamarketingPending;
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

RVD - Redundant Variable Declaration

Criticality	Minor / Informative
Location	Token.sol#L25
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The contract declares certain variables that are not used in a meaningful way by the contract. As a result, these variables are redundant.

```
uint16[3] public buddhamarketingFees;
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	Token.sol#L123
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.

```
isExcludedFromFees[account] = isExcluded;
```

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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	Token.sol#L34,38,39,40,67,87,102,111
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.

```
mapping (address => bool) public AMMPairs
event buddhamarketingAddressUpdated(address buddhamarketingAddress);
event buddhamarketingFeesUpdated(uint16 buyFee, uint16 sellFee, uint16
transferFee);
event buddhamarketingFeeSent(address recipient, uint256 amount);
address _router
uint16 _swapThresholdRatio
address _newAddress
uint16 _transferFee
uint16 _sellFee
uint16 _buyFee
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	Token.sol#L147,150
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.

```
fees = amount * totalFees[txType] / 10000
_buddhamarketingPending += fees * buddhamarketingFees[txType] /
totalFees[txType]
```

Recommendation

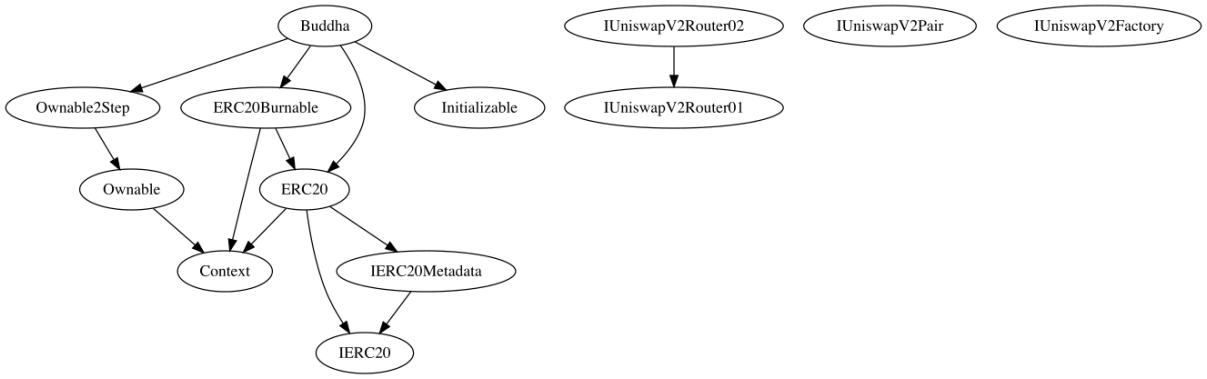
To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Buddha	Implementation	ERC20, ERC20Burnable, Ownable2Step, Initializable		
		Public	✓	ERC20
	initialize	External	✓	initializer
		External	Payable	-
	decimals	Public		-
	_swapTokensForCoin	Private	✓	
	updateSwapThreshold	Public	✓	onlyOwner
	getSwapThresholdAmount	Public		-
	getAllPending	Public		-
	buddhamarketingAddressSetup	Public	✓	onlyOwner
	buddhamarketingFeesSetup	Public	✓	onlyOwner
	excludeFromFees	Public	✓	onlyOwner
	_transfer	Internal	✓	
	_updateRouterV2	Private	✓	
	setAMMPair	External	✓	onlyOwner
	_setAMMPair	Private	✓	

	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	

Inheritance Graph



Flow Graph

Summary

Buddha contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. Buddha is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues. The contract Owner can access some admin functions that can not be used in a malicious way to disturb the users' transactions. 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