Cyberscope

Audit Report

Burnmeme

January 2023

SHA256 4cf7bc689ab8dee40c824283b9c0e831b063d6952b4fc0de783754da7fad4572

Audited by © cyberscope

Table of Contents

Review

Audit Updates

Initial Audit	26 Jan 2023

Source Files

Filename	SHA256
BurnMeme.sol	4cf7bc689ab8dee40c824283b9c0e831b063d6952b4fc0de783754da7fad4572

Introduction

Roles

Owner

- renounceOwnership()
- transferOwnership()
- rescueFunds()
- rescueTokens()
- togglePause()
- setMaxUserLimit()
- setMaintainenceFee()
- setDeadWalletFee()
- setToken()
- setSigner()
- setMaintainenceWallet()
- setDeadWallet()
- setmem()

Signer

- setTopThreeWinner()
- updateUserReward()
- randomPicker()

User

- deposit()
- withdraw()
- getUserLength()
- getTopThreeWinner()

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	OTUT	Transfers User's Tokens	Unresolved
●	WSR	Winner Selection Randomization	Unresolved
●	PTAI	Potential Transfer Amount Inconsistency	Unresolved
●	RV	Randomization Vulnerability	Unresolved
●	MVN	Misleading Variables Naming	Unresolved
●	MC	Missing Check	Unresolved
●	PFI	Potential Fund Insufficiency	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
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved

●	L19	Stable Compiler Version	Unresolved

OTUT - Transfers User's Tokens

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

Description

The contract owner has the authority to transfer the balance of a user's contract to the owner's contract. The owner may take advantage of it by calling the `rescueTokens` function.

```
function rescueTokens(IERC20 _token, address _recipient, uint256 _amount) external
onlyOwner {
    _token.safeTransfer(_recipient,_amount);
}
```

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. That risk can be prevented by temporarily locking the contract or renouncing ownership.

WSR - Winner Selection Randomization

Criticality	Minor / Informative
Location	BurnMeme.sol#L645
Status	Unresolved

Description

The contract does not select the top three winners randomly. Instead, the addresses are being passed as arguments to the `setTopThreeWinner` function. As a result, the signer can manipulate the winners by passing any addresses as arguments.

```
function setTopThreeWinner(address _first, address _second, address _third) external
onlySigner {
    require(_first != address(0) && _second != address(0) && _third != address(0),
"error: zero values");
    FirstWinner = _first;
    SecondWinner = _second;
    ThirdWinner = _third;
}
```

Recommendation

The team is advised to modify the `randomPicker` function to private instead of external and select each winner by calling it in the `setTopThreeWinner` function. This ensures that the winners are picked randomly.

PTAI - Potential Transfer Amount Inconsistency

Criticality	Minor / Informative
Location	BurnMeme.sol#L595,620,624
Status	Unresolved

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

Since the `XDC` token address can be modified by the owner, the amount that is sent may not be the same as the amount is added to `userRecord[_to]._amount` as an example, if the `XDC` token implements a tax or fee mechanism.

```
XDC.safeTransferFrom(_to, address(this), _amount);
...
XDC.safeTransfer(MaintainenceWallet, toMaintainence);
...
XDC.safeTransfer(DeadWallet, toDead);
```

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
```

RV - Randomization Vulnerability

Criticality	Minor / Informative
Location	BurnMeme.sol#L687
Status	Unresolved

Description

The contract is using an on-chain technique in order to determine random numbers. The blockchain runtime environment is fully deterministic, as a result, the pseudo-random numbers could be predicted.

```
function randomNumberGenerator(uint256 _upto) internal view returns(uint256){
    uint256 seed = uint256(keccak256(abi.encodePacked(
        block.timestamp + block.difficulty +
        ((uint256(keccak256(abi.encodePacked(block.coinbase)))) / (block.timestamp))
+
        block.gaslimit +
        ((uint256(keccak256(abi.encodePacked(msg.sender)))) / (block.timestamp)) +
        block.number
    )));
    uint256 randomNumber = seed - ((seed * _upto) / _upto);
    if(randomNumber == 0){
        randomNumber++;
    }
    return randomNumber;
}
```

Recommendation

The contract could use an advanced randomization technique that guarantees an acceptable randomization factor. For instance, the Chainlink VRF (Verifiable Random Function). https://docs.chain.link/docs/chainlink-vrf

MVN - Misleading Variables Naming

Criticality	Minor / Informative
Location	BurnMeme.sol#L756
Status	Unresolved

Description

Variables can have misleading names if their names do not accurately reflect the value they contain or the purpose they serve. The contract uses some variable names that are too generic or do not clearly convey the information stored in the variable. Misleading variable names can lead to confusion, making the code more difficult to read and understand. A variable called `DeadWallet` should represent the dead address. Instead, the contract uses this variable with the constraint that it must not be the dead address.

```
function setDeadWallet(address _address) external onlyOwner returns(address
deadWallet) {
    require(_address != address(0), "error: zero address");
    DeadWallet = _address;
    deadWallet = DeadWallet;
}
```

Recommendation

It's always a good practice for the contract to contain variable names that are specific and descriptive. The team is advised to keep in mind the readability of the code.

MC - Missing Check

Criticality	Minor / Informative
Location	BurnMeme.sol#L617
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues. The contract is transferring funds to a `MaintainenceWallet` and a `DeadWallet`. If the sum of `MaintainenceFee` and `DeadFee` is greater than the `PERCENT_DIVIDER`, the calculated amounts will be incorrect.

```
function deductTax(uint256 _beforeTax) internal returns(uint256 _afterTax){
    /*4% goes to Maintainence Wallet*/
    uint256 toMaintainence = (_beforeTax * MaintainenceFee) / PERCENT_DIVIDER;
    XDC.safeTransfer(MaintainenceWallet, toMaintainence);

    /*30% goes to Dead Wallet*/
    uint256 toDead = (_beforeTax * DeadFee) / PERCENT_DIVIDER;
    XDC.safeTransfer(DeadWallet, toDead);

    _afterTax = _beforeTax - (toMaintainence + toDead);
}
```

Recommendation

The team is advised to properly check the variables according to the required specifications. A recommendation would be to add the following code at the start of the `deductTax` function or at both the `setMaintainenceFee` and `setDeadWalletFee` functions:

```
require(MaintainenceFee + DeadFee <= PERCENT_DIVIDER, "The sum of fees cannot be greater than the PERCENT_DIVISOR")
```

PFI - Potential Fund Insufficiency

Criticality	Minor / Informative
Location	BurnMeme.sol#L652
Status	Unresolved

Description

If a smart contract is attempting to make a transfer of funds (such as with the `transfer()` or `send()` function) and the account sending the funds does not have enough to complete the transfer, the transaction will fail and the smart contract will throw an exception. This is known as an "insufficient funds" error. The contract can increase the user's reward amount by calling the `updateUserReward` function. The `XDC` token may not have enough funds and, as a result, the withdrawal may fail.

```
function updateUserReward(address _user, uint256 _reward) external onlySigner
returns(uint256 rewardAfterUpdate){
    userRecord[_user].totalWithdrawables += _reward;
    rewardAfterUpdate = userRecord[_user].totalWithdrawables;
}
```

Recommendation

The team is advised to check if the `XDC` token has enough funds before increasing the user's rewards.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	BurnMeme.sol#L549
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.

```
uint256 public PERCENT_DIVIDER = 1e4
```

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	BurnMeme.sol#L309,539,541,544,547,548,549,551,552,553,555,556,557,562,569,570,587,617,645,652,687,712,721,726,731,737,742,750,756,762
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.

```
function DOMAIN_SEPARATOR() external view returns (bytes32);
string public MEME
IERC20 public XDC
uint256 public MAX_USER_LIMIT = 500000
uint256 public MaintainenceFee
uint256 public DeadFee
uint256 public PERCENT_DIVIDER = 1e4
address public MaintainenceWallet
address public DeadWallet
address public Signer
address internal FirstWinner
address internal SecondWinner
address internal ThirdWinner

...
```

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	BurnMeme.sol#L723,728,733
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.

```
MAX_USER_LIMIT = _newLimit
MaintainenceFee = _fee
DeadFee = _fee
```

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	BurnMeme.sol#L68,93,122,149,159,174,184,223,430,441,446,455
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 sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

        (bool success, ) = recipient.call{value: amount}("");
        require(success, "Address: unable to send value, recipient may have
reverted");
    }

function functionCall(address target, bytes memory data) internal returns (bytes
memory) {
        return functionCallWithValue(target, data, 0, "Address: low-level call
failed");
    }

function functionCallWithValue(address target, bytes memory data, uint256 value)
internal returns (bytes memory) {
        return functionCallWithValue(target, data, value, "Address: low-level call
with value failed");
    }

...
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	BurnMeme.sol#L584
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
Signer = _signer
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	BurnMeme.sol#L240
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 {
                let returndata_size := mload(returndata)
                revert(add(32, returndata), returndata_size)
            }
```

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	BurnMeme.sol#L2
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;
```

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.

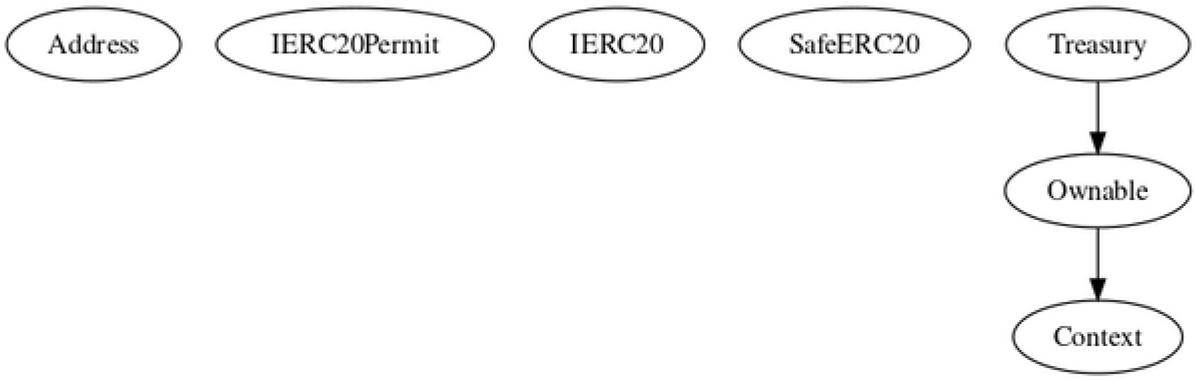
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	verifyCallResultFromTarget	Internal		
	verifyCallResult	Internal		
	_revert	Private		
IERC20Permit	Interface			
	permit	External	✓	-
	nonces	External		-
	DOMAIN_SEPARATOR	External		-
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-

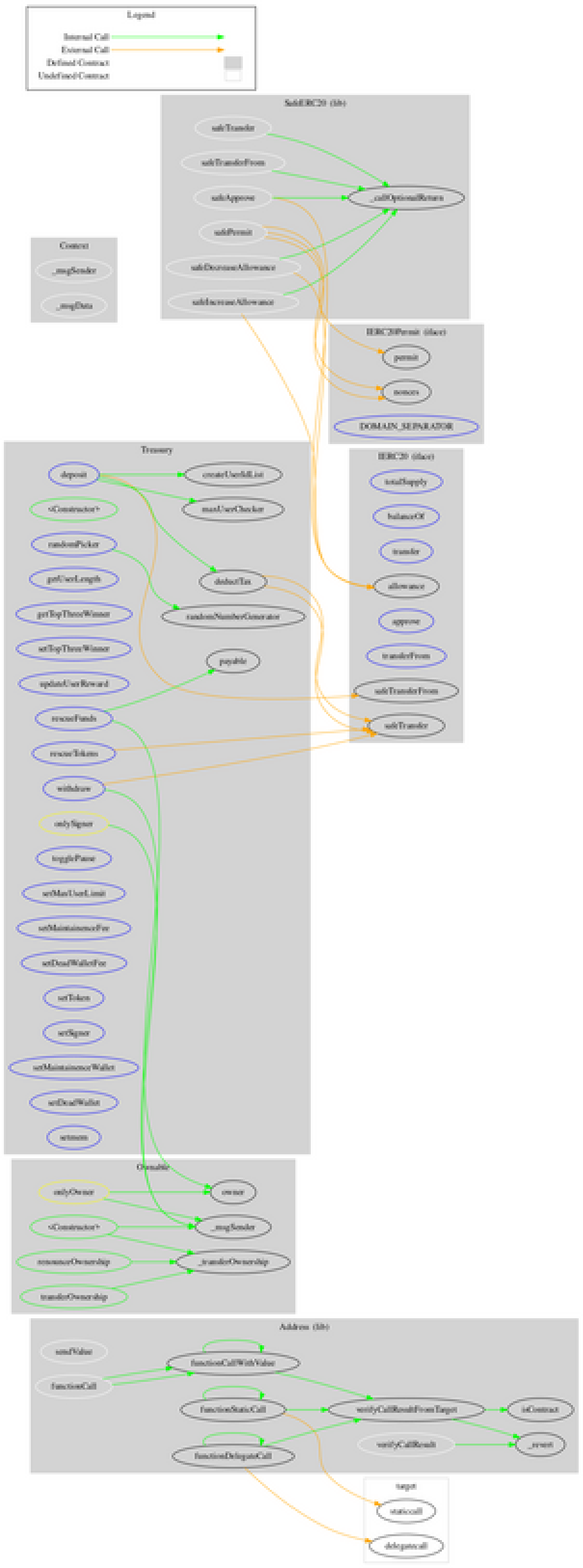
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
SafeERC20	Library			
	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeApprove	Internal	✓	
	safeIncreaseAllowance	Internal	✓	
	safeDecreaseAllowance	Internal	✓	
	safePermit	Internal	✓	
	_callOptionalReturn	Private	✓	
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
Treasury	Implementation	Ownable		
		Public	✓	-
	deposit	External	✓	-
	deductTax	Internal	✓	
	maxUserChecker	Internal		

	getUserLength	External		-
	getTopThreeWinner	External		-
	setTopThreeWinner	External	✓	onlySigner
	updateUserReward	External	✓	onlySigner
	withdraw	External	✓	-
	randomPicker	External		onlySigner
	createUserIdList	Internal	✓	
	randomNumberGenerator	Internal		
	rescueFunds	External	✓	onlyOwner
	rescueTokens	External	✓	onlyOwner
	togglePause	External	✓	onlyOwner
	setMaxUserLimit	External	✓	onlyOwner
	setMaintainenceFee	External	✓	onlyOwner
	setDeadWalletFee	External	✓	onlyOwner
	setToken	External	✓	onlyOwner
	setSigner	External	✓	onlyOwner
	setMaintainenceWallet	External	✓	onlyOwner
	setDeadWallet	External	✓	onlyOwner
	setmem	External	✓	onlyOwner

Inheritance Graph



Flow Graph



Summary

Burnmeme contract implements a staking mechanism. This audit investigates security issues, business logic concerns and potential improvements. Additionally, the contract owner has the authority to prevent users from withdrawing, by enabling the pause variable anytime.

Disclaimer

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