Cyberscope

Audit Report

Burnmeme

January 2023

SHA256 7df4278691270aa705f545911b3c5dbbb74d381c08da98e7705c35bdcd60e1af

Audited by © cyberscope

Table of Contents

Review

Audit Updates

Initial Audit	26 Jan 2023 https://github.com/cyberscope-io/audits/tree/main/brnmeme/v1/audit.pdf
Corrected Phase 2	30 Jan 2023

Source Files

Filename	SHA256
BurnMeme.sol	7df4278691270aa705f545911b3c5dbbb74d381c08da98e7705c35bdcd60e1af

Introduction

Roles

Owner

- `function renounceOwnership()`
- `function transferOwnership(address newOwner)`
- `function setMaxUserLimit(uint _newLimit)`
- `function setMaintainenceFee(uint256 _fee)`
- `function setDeadWalletFee(uint256 _fee)`
- `function setToken(address _token)`
- `function setSigner(address _address)`
- `function setMaintainenceWallet(address _address)`
- `function setDeadWallet(address _address)`
- `function setmem(string memory _str)`

Signer

- `function setTopThreeWinner(address _first, address _second, address _third)`
- `function updateUserReward(address _user, uint256 _reward)`
- `function randomPicker(uint256 limit)`

User

- `function deposit(uint _amount)`
- `function withdraw()`
- `function getUserLength()`
- `function getTopThreeWinner()`

Rewards Amount

During the withdrawal process, if the contract's funds are not sufficient to cover the rewarded amount, the contract will transfer to the user the remaining balance. As a result, the contract will assume that the entire amount has been credited to the user, but the user will receive a part of it.

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	WSR	Winner Selection Randomization	Unresolved
●	RV	Randomization Vulnerability	Unresolved
●	MVN	Misleading Variables Naming	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved

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.

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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	BurnMeme.sol#L309,543,545,548,551,552,555,556,557,559,560,561,565,572,573,599,633,661,668,709,732,737,745,754,761,769,777,785
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
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#L734
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
```

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.

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.

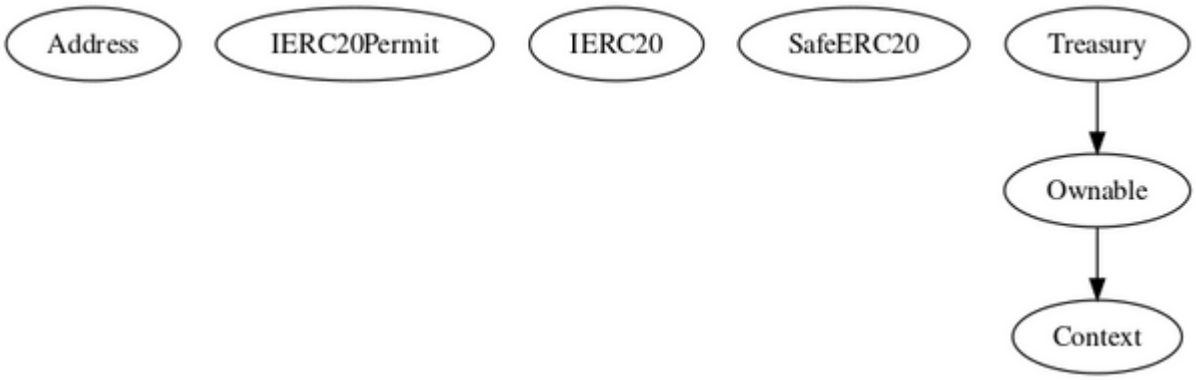
Functions Analysis

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		
	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


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
SafeERC20 (lib)
safeTransfer
safeTransferFrom
safeApprove
safeIncreaseAllowance
safePermit
safeDecreaseAllowance
_callOptionalReturn
Context
_msgSender
_msgData
IERC20Permit (iface)
permit
nonces
DOMAIN_SEPARATOR
IERC20 (iface)
totalSupply
allowance
transfer
safeTransfer
approve
transferFrom
safeTransferFrom
balanceOf
Treasury
deposit
createUserIdList
maxUserChecker
deductTax
randomNumberGenerator
<Constructor>
randomPicker
getUserLength
getTopThrowWinner
setTopThrowWinner
updateUserReward
withdraw
setToken
setMaxUserLimit
setMaintainenceFee
setDeadWalletFee
onlySigner
setSigner
setMaintainenceWallet
setDeadWallet
setmem
_token
isContract
Ownable
<Constructor>
_msgSender
onlyOwner
owner
renounceOwnership
_transferOwnership
transferOwnership
Address (lib)
sendValue
functionCall
functionCallWithValue
functionStaticCall
functionDelegateCall
verifyCallResultFromTarget
verifyCallResult
isContract
_revert
target
staticcall
delegatecall

Summary

Burnmeme contract implements a staking mechanism. This audit investigates security issues, business logic concerns and potential improvements.

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