Cyberscope

Audit Report
Brainers

February 2024

Network	ETH
Address	0x5044d567f7b30891639d982a05726a6bfe8bae6a
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	Mint Tokens	Unresolved
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	IVC	Incorect Vesting Calulcation	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	MCM	Misleading Comment Message	Unresolved
●	RVU	Redundant Variables Usage	Unresolved
●	L13	Divide before Multiply Operation	Unresolved

Table of Contents

Review

Contract Name	Brainers
Compiler Version	v0.8.24+commit.e11b9ed9
Optimization	200 runs
Explorer	https://etherscan.io/address/0x5044d567f7b30891639d982a05726a6bfe8bae6a
Address	0x5044d567f7b30891639d982a05726a6bfe8bae6a
Network	ETH
Symbol	BRAINERS
Decimals	18
Total Supply	371,000,000
Badge Eligibility	Must Fix Criticals

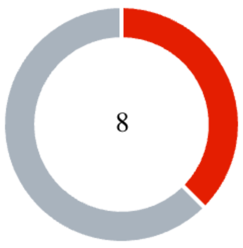
Audit Updates

Initial Audit	14 Feb 2024

Source Files

Filename	SHA256
Brainers.sol	9651deecb01c1f3651f52428f732b622aad7649d589f9ff71f66a10f8c6daa88

Findings Breakdown



●	Critical	3
●	Medium	0
●	Minor / Informative	5

	Severity	Unresolved	Acknowledged	Resolved	Other
●	Critical	3	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	5	0	0	0

MT - Mint Tokens

Criticality	Critical
Location	Brainers.sol#L128
Status	Unresolved

Description

The total supply of a token is the total number of tokens that have been created, while the balances of individual accounts represent the number of tokens that an account owns. The total supply and the balances of individual accounts are two separate concepts that are managed by different variables in a smart contract. These two entities should be equal to each other.

In the contract, the amount that is added to the total supply does not equal the amount that is added to the balances. As a result, the sum of balances is diverse from the total supply.

Specifically, the contract is designed to implement a token release functionality for specific addresses (`_teamAddress` and `_socialNetworkAddress`) after a predetermined time. However, the balances for `_teamAddress` and `_socialNetworkAddress` are set within the contract's constructor, based on predefined supplies. Subsequently, the release functions (`releaseTeamSupply` and `releaseSocialSupply`) aim to increase the balances of these addresses further as part of the release process. This approach inadvertently inflates the balances without proper accounting for the initial setup. Moreover, during the execution of the `releaseSocialSupply` function, the contract decreases `_totalSupply` directly without deducting any specific balance, leading to a discrepancy in the token distribution logic. This inconsistency can result in an inaccurate reflection of the total supply and individual balances, potentially undermining the token's economic model and trust in the contract's management of token supplies.

```
_balances[_teamAddress] = _teamSupply;
...

function releaseTeamSupply() external onlyUnlockedTeam {
...
    _balances[_teamAddress] += releaseAmount;
...
    }
```

```
_balances[_socialNetworkAddress] = (_totalSupply *
socialPercentage) / 100;
...
    function releaseSocialSupply() external onlyOwner {
    ...
        _balances[_socialNetworkAddress] += releaseAmount;
        _totalSupply -= releaseAmount;
    ...
    }
```

Recommendation

The total supply and the balance variables are separate and independent from each other. The total supply represents the total number of tokens that have been created, while the balance mapping stores the number of tokens that each account owns. The sum of balances should always equal the total supply. It is recommended to reconsider the release functionality of both the `releaseTeamSupply` and `releaseSocialSupply` functions. If the intended functionality is to release a certain amount of tokens after a specific time, then the contract should adjust the specific balances of the corresponding addresses at the time of the release, ensuring that the initial allocations are considered in the total. Additionally, any modification to the total supply should be accurately reflected in the corresponding balance changes to maintain consistency in the token's ledger.Implementing these changes will ensure that the release process is transparent, predictable, and aligns with the intended token distribution strategy, thereby maintaining the integrity of the token's economic framework.

IVC - Incorrect Vesting Calculation

Criticality	Critical
Status	Unresolved

Description

The variable `_teamReleaseStartTime` is not initialized in the contract, as a result, every time that the expression `_teamReleaseStartTime += _teamReleaseInterval` is evaluated, it will add `30 days` to the `_teamReleaseStartTime` variable. This will result in a number that will always be less than the current timestamp. Hence, the invariant `_teamReleaseStartTime + _teamReleaseInterval <= block.timestamp` will always yield false values.

```
function releaseTeamSupply() external onlyUnlockedTeam {
    require(_teamReleaseStartTime + _teamReleaseInterval <=
block.timestamp, "Team supply release interval not reached yet");

    uint256 releaseAmount = _teamReleaseAmount;
    if (releaseAmount > _teamSupply) {
        releaseAmount = _teamSupply;
    }

    _balances[_teamAddress] += releaseAmount;
    _teamSupply -= releaseAmount;
    _teamReleaseStartTime += _teamReleaseInterval;

    emit Transfer(address(this), _teamAddress, releaseAmount);
}
```

Recommendation

The team is advised to take into consideration the expected business logic and adjust the implementation properly.

MEE - Missing Events Emission

Criticality	Critical
Location	Brainers.sol#L120,135
Status	Unresolved

Description

The contract emits transfer events that do not follow the actual balance transitions. As a result, it produces misleading information to the users. In the following examples, the amount is minted to the specific addresses, but the event describes that the tokens are transferred from the contract address to the corresponding wallet.

```
_balances[_socialNetworkAddress] += releaseAmount;
_totalSupply -= releaseAmount;
_socialReleaseStartTime += _socialReleaseInterval;

emit Transfer(address(this), _socialNetworkAddress, releaseAmount);

//...

_balances[_teamAddress] += releaseAmount;
_teamSupply -= releaseAmount;
_teamReleaseStartTime += _teamReleaseInterval;

emit Transfer(address(this), _teamAddress, releaseAmount);
```

Recommendation

It is recommended to emit the events that describe the actual transition of the tokens. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	Brainers.sol#L53,61,62,63,64,66,67
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
_totalSupply
_teamSupply
_teamUnlockTime
_teamReleaseAmount
_teamReleaseInterval
_socialReleaseInterval
_socialReleaseStartTime
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

MCM - Misleading Comment Message

Criticality	Minor / Informative
Location	Brainers.sol#L138
Status	Unresolved

Description

The contract is using misleading comment messages. These comment messages do not accurately reflect the actual implementation, making it difficult to understand the source code. As a result, the users will not comprehend the source code's actual implementation.

Specifically, the contract is designed to allocate a specific portion of the total supply as a release amount, with the implementation setting the `releaseAmount` to represent 80% of the totalSupply. However, there is a significant discrepancy between the code's functionality and the accompanying comment. The comment incorrectly states that the calculation represents `10% of Brainers Social allocation,` which is misleading given that the actual code calculates 80% of the total supply. This inconsistency between the code and its documentation can lead to confusion for stakeholders, potentially misinforming them about the contract's intended logic and the distribution strategy of the tokens.

```
 uint256 releaseAmount = (_totalSupply * 80) / 1000; // 10% of
Brainers Social allocation
```

Recommendation

The team is advised to carefully review the comment in order to reflect the actual implementation. To improve code readability, the team should use more specific and descriptive comment messages. It is recommended to modify the comment to accurately reflect the code's implementation. If the intended release amount is indeed 80% of the total supply, the comment should be corrected to state `// 80% of the total supply` to eliminate any ambiguity and ensure clarity for all parties reviewing the code. Ensuring that comments accurately describe the code's functionality is crucial for maintaining transparency, preventing misunderstandings that could affect the contract's operation and the project's integrity.

RVU - Redundant Variables Usage

Criticality	Minor / Informative
Location	Brainers.sol#L60,75
Status	Unresolved

Description

The contract redundantly uses separate variables to set the balance of the contract, specifically for allocating percentages of the total supply to different purposes such as marketing, development, investor relations, and future development. Each of these allocations is defined by its own variable (`marketingPercentage` , `developmentPercentage` , `investorPercentage` , `futureDevelopmentPercentage`), and the total balance for the contract is calculated by summing these percentages. While this approach provides clarity in terms of allocation breakdown, it introduces unnecessary complexity if the functionality of the contract does not leverage these distinct values for separate operations or if the differentiation does not serve a specific, functional purpose.

```
uint256 marketingPercentage = 5;
uint256 developmentPercentage = 2;
uint256 investorPercentage = 2;
uint256 futureDevelopmentPercentage = 8;
...
_balances[address(this)] = (_totalSupply * (marketingPercentage +
developmentPercentage + investorPercentage +
futureDevelopmentPercentage)) / 100;
```

Recommendation

It is recommended to consolidate these allocations into a single variable if the intended functionality does not require utilizing the different variables' values for distinct operations. This can be achieved by defining a single `totalAllocationPercentage` that represents the sum of all intended allocations. If differentiation is necessary for documentation or clarity purposes, comments can be used to detail the breakdown of the total allocation.

Simplifying the variable structure in this manner will enhance the contract's readability and maintainability, reducing the potential for errors and making the codebase more efficient. This approach streamlines the contract's logic without sacrificing the transparency of fund allocation.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	Brainers.sol#L61,63
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.

```
_teamSupply = (_totalSupply * teamPercentage) / 100
_teamReleaseAmount = (_teamSupply * 5) / 100
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	Brainers.sol#L12
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 ^0.8.11;
```

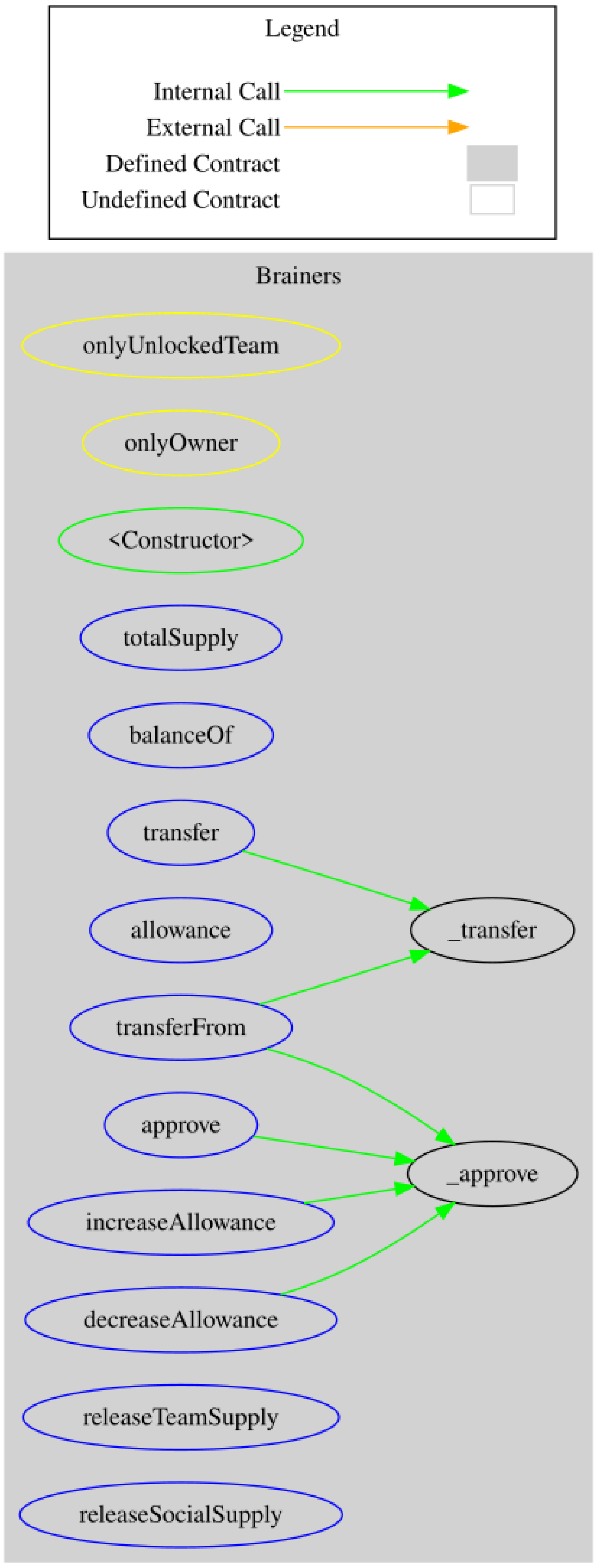
Recommendation

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**
Brainers	Implementation			
		Public	✓	-
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
	increaseAllowance	External	✓	-
	decreaseAllowance	External	✓	-
	releaseTeamSupply	External	✓	onlyUnlockedTeam
	releaseSocialSupply	External	✓	onlyOwner
	_transfer	Internal	✓	
	_approve	Internal	✓	

Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Brainers
onlyUnlockedTeam
onlyOwner
<Constructor>
totalSupply
balanceOf
transfer
allowance
_transfer
transferFrom
approve
_approve
increaseAllowance
decreaseAllowance
releaseTeamSupply
releaseSocialSupply

Summary

Brainers contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. Brainers is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error but some critical issues.

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