THE ARCADIA GROUP

AUDIT

arcadiamgroup.com

Table of Contents

Executive Summary

A Representative Party of the MyCryptoPlay ("MCP") engaged The Arcadia Group ("Arcadia"), a software development, research, and security company, to conduct a review of the following MCPToken smart contracts on the MCPToken contract address :

0x2186Ecb39f1B765bA7d78f1C43c2E9D7Fc0C1eca (Mainnet)

MCPToken.sol

ERC20Vestable.sol

Findings

1. Unclear definition of totalSupply

MCP-1
Severity: Code Clarity
Impact: Code Clarity

Contract: MCPToken
Category: Information
Finding Type: Dynamic
Lines: 513-535

The contract states that the total supply will be sent to the tokenOwnerAddress but eventually the logic inside the constructor doesn't match with the expectation.

The constructor mint a 'totalSupply' to the 'tokenOwnerAddress' .

Functions from line 528 to line 536 actually mint new tokens so that the actual total Supply is totalSupply+ development amount + teamReserved amount + mcpStaking amount + marketing amount + privateSaleAllocation amount, which leads to a new totalSupply equal to totalSupply to owner + 26M tokens.

Part of this 26M supply is immediately minted like in the cases of privateSale and marketing where 100% is minted, while in case of development and mcpStaking, 1/25 of their respective amount will be minted immediately, the rest will be created in the long run each time _grantFunds() will be invoked by claimFunds(). TeamReserved will be also minted over time with the same process.

The variable names like totalSupply and the comments should be renamed properly because can lead to misunderstanding from external readers. One potential naming clarification is as circulatingSupply.

```
/**
* @dev Constructor.
* @param name name of the token
* @param symbol symbol of the token, 3-4 chars is recommended
* @param totalSupply total supply of tokens in lowest units (depending
on decimals)
* @param tokenOwnerAddress address that gets 100% of token supply
*/
constructor(string memory name, string memory symbol, uint256
totalSupply, address payable feeReceiver, address tokenOwnerAddress,
address development, address teamReserved, address mcpStaking, address
marketing, address privateSale) public payable {
_name = name;
_symbol = symbol;
_decimals = 18;

// set tokenOwnerAddress as owner of all tokens
_mint(tokenOwnerAddress, totalSupply.mul(10**uint256(_decimals)));
// allocation and vesting of various funds
_addBeneficiary(development, 6000000, 25, 28 days);
_addBeneficiaryWithoutInitialGrant(teamReserved, 6000000, 25, 730
days);
_addBeneficiary(mcpStaking, 8000000, 25, 28 days);

//Transfering Marketing and Private sale funds to their respective
addresses
uint256 marketingFunds = 4000000;
uint256 privateSaleAllocation = 2000000;
_mint(marketing,marketingFunds.mul(10**uint256(_decimals)));
_mint(privateSale,privateSaleAllocation.mul(10**uint256(_decimals)));

// pay the service fee for contract deployment
feeReceiver.transfer(msg.value);
}
```

Resolution

While the contract function cannot be clarified in production due to the MCP Contract being deployed, in future iterations the naming should be corrected.

Additionally, the below economics have been clarified with the team to be the basis for a more accurate totalSupply

```
Initial issuance of 14m with subsequent issuances of
Staking - 4% minted every 28 days. Total - 8m
Development - 4% minted every 28 days Total - 6m
Team - 4% minted after 2 years for every 90 days - 6m
Marketing - minted on deployment Total - 4m
Private Sale - minted on deployment total - 2m
totalSupply= 14m + 26m = 40m
```

2. Rounding issue when dividing claimable amount

MCP-2
Severity: Informational
Impact: Reuse

Contract: ERC20Vestable
Category: Informational
Finding Type: Dynamic
Lines: 475

The result of ‘amount.div(divide)’ could lead to rounding issues (in this case the math is hardcoded in the MCPToken contract so we know in advance it won’t have any rounding issue with the actual values) but if reused with other contracts or different values then rounding issue should be taken into account to avoid that some tokens could be unclaimable.

```
function _addBeneficiary(address beneficiary, uint256 amount, uint8
divide, uint256 claimFrequency)
internal
{
_vestingAllowances[beneficiary] = amount;
_claimFrequency[beneficiary] = claimFrequency;
_claimAmounts[beneficiary] = amount.div(divide);
_lastClaimed[beneficiary] = block.timestamp;
// beneficiary gets first division of funds immediately
_grantFunds(beneficiary);
}
```

Disclaimer

While best efforts and precautions have been taken in the preparation of this document, The Arcadia Group and the Authors assume no responsibility for errors, omissions, or damages resulting from the use of the provided information. Additionally, Arcadia would like to emphasize that the use of Arcadia's services does not guarantee the security of a smart contract or set of smart contracts and does not guarantee against attacks. One audit on its own is not enough for a project to be considered secure; that categorization can only be earned through extensive peer review and battle testing over an extended period.

Document Information

Title	MCPToken Custom ERC20 Audit
Client	MyCryptoPlay
Auditor(s)	Andrea Burzi
Reviewed by	Joel Farris
Approved by	Rasikh Morani
Contact Details	Rasikh Morani (972) 543-3886 rasikh@arcadiamgroup.com https://t.me/thearcadiagroup