Smart Contract Security Audit Report

The SlowMist Security Team received the team's application for smart contract security audit of the GM NFT - CyberV on 2024.04.28. The following are the details and results of this smart contract security audit:

Token Name :

GM NFT - CyberV

The contract address :

https://etherscan.io/address/0x67ce4afa08ebf2d6d1f31737cc5d54ff116205e9

The audit items and results :

(Other unknown security vulnerabilities are not included in the audit responsibility scope)

NO.	Audit Items	Result
1	Replay Vulnerability	Passed
2	Denial of Service Vulnerability	Passed
3	Race Conditions Vulnerability	Passed
4	Authority Control Vulnerability Audit	Passed
5	Integer Overflow and Underflow Vulnerability	Passed
6	Gas Optimization Audit	Passed
7	Design Logic Audit	Passed
8	Uninitialized Storage Pointers Vulnerability	Passed
9	Arithmetic Accuracy Deviation Vulnerability	Passed
10	"False top-up" Vulnerability	Passed
11	Malicious Event Log Audit	Passed
12	Scoping and Declarations Audit	Passed
13	Safety Design Audit	Passed
14	Non-privacy/Non-dark Coin Audit	Passed

Audit Result : Passed

Audit Number : 0X002404300002

Audit Date : 2024.04.28 - 2024.04.30

Audit Team : SlowMist Security Team

Summary conclusion : This is an ERC721 Token contract and does not contain the dark coin section. The total amount and unit price of NFT remain unchangeable. the contract does not have the Overflow and the Race Conditions issue. During the audit, we found the following information:

1. Only The owner role can set the signer address through the setSigner function.
2. Only The owner role can set the base token url through the setBaseURI function.
3. Only The owner role can set the time for whitelist mint through the setWhiteListMintTimeFrame function.
4. Only The owner role can set the time for public mint through the setPublicMintTimeFrame function.
5. The owner role can transfer any tokens in the contract through the withdraw function.

The source code:

```
// SPDX-License-Identifier: LGPL-3.0-only
//SlowMist// The contract does not have the Overflow and the Race Conditions issue.
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
import "@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol";
import "@openzeppelin/contracts/access/Ownable2Step.sol";
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
import "@openzeppelin/contracts/utils/Strings.sol";
import "@openzeppelin/contracts/utils/cryptography/ECDSA.sol";

contract CyberV is ERC721Enumerable, Ownable2Step, ReentrancyGuard {
    using SafeERC20 for IERC20;

    // base url
    string private _baseTokenURI;
    // signer
    address public signer;
    // total count
    uint256 public totalCount = 2077;
    // mint price
```

```
    uint256 public price = 127*10**15;
    // mint time stamp
    uint256 public whitelistStartTime;
    uint256 public whitelistEndTime;
    uint256 public publicStartTime;
    uint256 public publicEndTime;
    // count per address
    uint8 public countPerAddress = 2;

    // owner -> count
    mapping(address => uint256) public mintStats;

    constructor(string memory baseTokenURI, address _owner, address _signer)
ERC721("CyberV", "CyberV") {
        _baseTokenURI = baseTokenURI;
        _transferOwnership(_owner);
        signer = _signer;
    }

    function getMintPeriod() public view returns(uint256) {
        if (block.timestamp >= whitelistStartTime && block.timestamp <=
whitelistEndTime) {
            return 1;
        }
        else if (block.timestamp >= publicStartTime && block.timestamp <=
publicEndTime) {
            return 2;
        }
        return 0;
    }

    function mint(uint256 _count, bytes calldata _signature) external payable
nonReentrant {
        uint256 mintPeriod = getMintPeriod();
        require(
            mintPeriod > 0,
            "CyberV: mint not started or already closed"
        );

        require(
            _count > 0 && _count <= countPerAddress - mintStats[msg.sender],
            "CyberV: invalid count"
        );

        require(
            totalSupply() < totalCount,
            "CyberV: exceed total available supply"
        );
```

```
        require(msg.value == price * _count, "CyberV: invalid eth value");

        if (mintPeriod == 1) {
            bytes32 message = ECDSA.toEthSignedMessageHash(
              keccak256(
                abi.encodePacked(_getChainID(), address(this), msg.sender)
              )
            );
            require(
              ECDSA.recover(message, _signature) == signer,
              "CyberV: invalid signature"
            );
        }

        for(uint i=0; i<_count; i++) {
            mintStats[msg.sender] = mintStats[msg.sender] + 1;
            _safeMint(msg.sender, totalSupply() + 1);
        }
    }

    function tokenURI(uint256 _tokenId)
        public
        view
        virtual
        override
        returns (string memory)
    {
        require(_exists(_tokenId), "CyberV: nonexistent token id");

        return
            string(
                abi.encodePacked(
                    _baseTokenURI,
                    "/",
                    Strings.toString(_tokenId)
                )
            );
    }

    //SlowMist// Only The owner role can set the signer address.
    function setSigner(address _signer) external onlyOwner {
        signer = _signer;
    }

    //SlowMist// Only The owner role can set the base token url.
    function setBaseURI(string memory baseTokenURI) external onlyOwner {
        _baseTokenURI = baseTokenURI;
    }
```

```
    //SlowMist// Only The owner role can set the time for whitelist mint.
    function setWhiteListMintTimeFrame(uint256 _whitelistStartTime, uint256
_whitelistEndTime) external onlyOwner {
        whitelistStartTime = _whitelistStartTime;
        whitelistEndTime = _whitelistEndTime;
    }

    //SlowMist// Only The owner role can set the time for public mint.
    function setPublicMintTimeFrame(uint256 _publicStartTime, uint256 _publicEndTime)
external onlyOwner {
        publicStartTime = _publicStartTime;
        publicEndTime = _publicEndTime;
    }

    //SlowMist// The owner role can transfer any tokens in the contract through the
withdraw function.
    function withdraw(address _erc20) external onlyOwner {
        if (_erc20 == address(0)) {
            (bool sent, bytes memory data) = payable(owner()).call{value:
address(this).balance}("");
            require(sent, "Failed to withdraw Ether");
        } else {
            IERC20 token = IERC20(_erc20);
            token.safeTransfer(owner(), token.balanceOf(address(this)));
        }
    }

    function _getChainID() internal view returns (uint256) {
        uint256 id;
        assembly {
            id := chainid()
        }
        return id;
    }

    receive() external payable {}
}
```

Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist