Smart Contract Security Audit Report

[2021]

The SlowMist Security Team received the HOTCROSS team's application for smart contract security audit of the Hot Cross Token on 2021.06.29. The following are the details and results of this smart contract security audit:

Token Name :

Hot Cross Token

Project address :

https://github.com/hotcrosscom/hotcross-token-solidity

Commit: 26c0b3c62bc96ac205bde4c965cc65f5532c7cb8

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	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

NO.	Audit Items	Result
12	Scoping and Declarations Audit	Passed
13	Safety Design Audit	Passed

Audit Result : Passed

Audit Number : 0x002106290005

Audit Date : 2021.06.29 - 2021.06.29

Audit Team : SlowMist Security Team

Summary conclusion : This is a token contract that contains the Vesting and Factory section. The total amount of tokens in the contract remains unchangeable. The contract does not have the Overflow and the Race Conditions issue.

The source code:

Misc.sol

```
//SlowMist// The contract does not have the Overflow and the Race Conditions issue
// SPDX-License-Identifier: MIT
pragma solidity 0.8.3;

library Misc {
  /**
    * @dev Returns true if `account` is a contract.
    *
    * [IMPORTANT]
    * ====
    * It is unsafe to assume that an address for which this function returns
    * false is an externally-owned account (EOA) and not a contract.
    *
    * Among others, `isContract` will return false for the following
    * types of addresses:
    *
    *  - an externally-owned account
    *  - a contract in construction
    *  - an address where a contract will be created
```

```
     *  - an address where a contract lived, but was destroyed
     * ====
     */
  function isContract(address account) public view returns (bool) {
    // This method relies on extcodesize, which returns 0 for contracts in
    // construction, since the code is only stored at the end of the
    // constructor execution.

    uint256 size;
    // solhint-disable-next-line no-inline-assembly
    assembly { size := extcodesize(account) }
    return size > 0;
  }
}
```

Migrations.sol

```
//SlowMist// The contract does not have the Overflow and the Race Conditions issue
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.22 <0.9.0;

contract Migrations {
  address public owner = msg.sender;
  uint public last_completed_migration;

  modifier restricted() {
    require(
      msg.sender == owner,
      "This function is restricted to the contract's owner"
    );
    _;
  }

  function setCompleted(uint completed) public restricted {
    last_completed_migration = completed;
  }
}
```

HotCross.sol

```
//SlowMist// The contract does not have the Overflow and the Race Conditions issue
// SPDX-License-Identifier: MIT
```

```
pragma solidity 0.8.3;

import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract HotCrossToken is Ownable, ERC20 {
  uint256 constant MILLION = (10**6) * 10**uint256(18);

  constructor() ERC20('Hot Cross Token', 'HOTCROSS') {
    _mint(msg.sender, 500 * MILLION);
  }


  /**
   * Allows the owner of the contract to release tokens that were erronously sent to
this
   * ERC20 smart contract. This covers any mistakes from the end-user side
   * @param token the token that we want to withdraw
   * @param recipient the address that will receive the tokens
   * @param amount the amount of tokens
   */
  function tokenRescue(
    IERC20 token,
    address recipient,
    uint256 amount
  ) onlyOwner external {
    token.transfer(recipient, amount);
  }
}
```

CrossVesting.sol

```
//SlowMist// The contract does not have the Overflow and the Race Conditions issue
// SPDX-License-Identifier: MIT
pragma solidity 0.8.3;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "../libs/Misc.sol";

contract CrossVesting is Ownable {
  using SafeERC20 for ERC20;
```

```
  struct VestingInfo {
    uint256 startTime;
    uint256 vestingDuration;
    uint256 cliff;
    uint256 amount;
    uint256 totalClaimed;
    uint256 periodClaimed;
  }

  ERC20 public token;
  uint256 public totalVestingCount;
  mapping (address => VestingInfo) private beneficiaries;

  event BeneficiaryAdded(address indexed beneficiary);
  event TokensClaimed(address indexed beneficiary, uint256 amountClaimed);
  event VestingRevoked(address beneficiary, uint256 amountVested, uint256
amountNotVested);

  constructor(ERC20 _token) {
    require(Misc.isContract(address(_token)), "Invalid token address");
    token = _token;
  }

  function currentTime() private view returns(uint256) {
    return block.timestamp;
  }

  function getVestingInfo(address beneficiary) public view returns(VestingInfo
memory) {
    return beneficiaries[beneficiary];
  }

  /**
   * @notice Registers a new beneficiary address
   * @param beneficiary The beneficiary address
   * @param startTime The beggining of the vesting contract
   * @param amount The amount of the token locked in the vesting contract
   * @param vestingDuration Vesting duration calculated in seconds
   * @param cliff Vesting cliff calculated in seconds
   */
  function addBeneficiary(
    address beneficiary,
    uint256 startTime,
    uint256 vestingDuration,
    uint256 cliff,
```

```
    uint256 amount
) external onlyOwner {
    require(beneficiary != address(0), "Beneficiary cannot be zero address");
    require(beneficiaries[beneficiary].amount == 0, "Beneficiary already exists");

    beneficiaries[beneficiary] = VestingInfo({
        startTime: startTime,
        vestingDuration: vestingDuration,
        cliff: startTime + cliff,
        amount: amount,
        totalClaimed: 0,
        periodClaimed: 0
    });

    // Transfer the vested tokens under the control of the vesting contract
    token.safeTransferFrom(owner(), address(this), amount);

    emit BeneficiaryAdded(beneficiary);
}

/**
 * @notice Allows a beneficiary to claim their vested tokens.
 */
function release() external {
    uint256 periodToVest;
    uint256 amountToVest;

    (periodToVest, amountToVest) = getTokenReleaseInfo(msg.sender);

    if(amountToVest > 0) {
        VestingInfo storage vestingInfo = beneficiaries[msg.sender];
        vestingInfo.periodClaimed += periodToVest;
        vestingInfo.totalClaimed += amountToVest;

        token.safeTransfer(msg.sender, amountToVest);
        emit TokensClaimed(msg.sender, amountToVest);
    }
}

/**
 * @notice Terminate vesting transferring all vested tokens to the  beneficiary
 * and returning all non-vested tokens to the contract owner
 * @param beneficiary The beneficairy address
 */
function revoke(address beneficiary) external onlyOwner {
```

```
    VestingInfo memory vestingInfo = beneficiaries[beneficiary];
    uint256 periodToVest;
    uint256 amountToVest;
    (periodToVest, amountToVest) = getTokenReleaseInfo(beneficiary);
    uint256 amountNotVested = vestingInfo.amount - vestingInfo.totalClaimed -
amountToVest;

    // reset all values
    beneficiaries[beneficiary] = VestingInfo({
      startTime: 0,
      vestingDuration: 0,
      cliff: 0,
      amount: 0,
      totalClaimed: 0,
      periodClaimed: 0
    });

    token.safeTransfer(owner(), amountNotVested);
    token.safeTransfer(beneficiary, amountToVest);

    emit VestingRevoked(beneficiary, amountToVest, amountNotVested);
  }

  /**
   * @notice Calculate the vested and unclaimed months and tokens available to claim
   * Due to rounding errors once vesting duration is reached, returns the entire left
amount
   * @dev Returns (0, 0) if cliff has not been reached
   * @param beneficiary The beneficairy address
   * @return The vested and unclaimed months and tokens available to claim
   */
  function getTokenReleaseInfo(address beneficiary) private view returns (uint256,
uint256) {
    VestingInfo memory vestingInfo = beneficiaries[beneficiary];

    require(vestingInfo.totalClaimed < vestingInfo.amount, "Tokens fully claimed");

    // For vesting created with a future start date, that hasn't been reached, return
0, 0
    if (currentTime() < vestingInfo.cliff) {
      return (0, 0);
    }

    uint elapsedPeriod = currentTime() - vestingInfo.startTime;
    uint256 periodToVest = elapsedPeriod - vestingInfo.periodClaimed;
```

```
    // If over vesting duration, all tokens vested
    if(elapsedPeriod >= vestingInfo.vestingDuration) {
      uint256 amountToVest = vestingInfo.amount - vestingInfo.totalClaimed;
      return (periodToVest, amountToVest);
    } else {
      uint256 amountToVest = (periodToVest * vestingInfo.amount) /
vestingInfo.vestingDuration;
      return (periodToVest, amountToVest);
    }
  }
}
```

CrossVestingFactory.sol

```
// SPDX-License-Identifier: MIT
//SlowMist// The contract does not have the Overflow and the Race Conditions issue
pragma solidity 0.8.3;

import "./CrossVesting.sol";
import "@openzeppelin/contracts/utils/Create2.sol";

contract CrossVestingFactory {
  struct VestingContract {
    string name;
    address contractAddress;
  }

  VestingContract[] public vestingContracts;

  event CrossVestingDeployed(address indexed vestingAddress, uint256 vestingIndex,
address token);

  function deployCrossVesting(
    bytes32 salt,
    ERC20 token,
    string memory name
  ) external returns (address) {
    bytes memory bytecode = abi.encodePacked(
      type(CrossVesting).creationCode,
      abi.encode(token)
    );
```

```
    address addr = Create2.deploy(0, salt, bytecode);

    vestingContracts.push(VestingContract({
      name: name,
      contractAddress: addr
    }));

    CrossVesting vesting = CrossVesting(addr);

    vesting.transferOwnership(msg.sender);

    emit CrossVestingDeployed(addr, vestingContracts.length - 1, address(token));

    return addr;
  }

  function vestingsCount() public view returns(uint256) {
    return vestingContracts.length;
  }
}
```

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